现代生物技术前沿

INTRODUCTION TO COMPUTATIONAL MOLECULAR BIOLOGY

〔巴西〕J.塞图宝
J.梅丹尼斯 著

计算分子生物学导论

朱 浩等译

7

科学出版社
www.sciencep.com

VISIT…

LANZAROTE Caliente.com

现代生物技术前沿

INTRODUCTION TO COMPUTATIONAL MOLECULAR BIOLOGY

〔巴西〕J.塞图宝
J.梅丹尼斯 著

计算分子生物学导论

朱浩等译

華僑大學圖書館藏書
A1073927

科学出版社
北京

图字:01-2003-0482

内 容 简 介

分子生物学的迅速发展,产生了大量的数据,而其间的关系也日趋复杂。计算分子生物学就是处理这些数据的新学科。

本书主要是介绍分子生物学中具有代表性的计算问题以及某些求解这些问题的有效算法。具体包括:分子生物学的基本概念;两个重要的数学对象,即串和图,和算法的基本概念等;序列比较和经典的动态程序设计算法;DNA 片段组装技术;DNA 的物理作图问题和一种物理作图的近似算法及启发式;与种系发生树构造有关的一些数学问题和某些用于构造特定类型种系发生树的算法;用以研究 DNA 中序列差异的数学模型和用于 RNA 结构预测的动态程序设计法以及蛋白质比配方法;最后还介绍了 DNA 计算。

本书可供研究基因组学与分子生物学的生物学、数学、计算机科学等专业的科研人员、教师、研究生等参考。

João Setubal, João Meidanis

Introduction to Computational Molecular Biology. 1st ed.

Copyright © 1997 by Brooks/Cole Publishing Company, a division of Thomson Learning

ISBN 0-534-95262-3

图书在版编目(CIP)数据

计算分子生物学导论 /(巴西)塞图宝,(巴西)梅丹尼斯著;朱浩等译 .
—北京:科学出版社,2003.8

(现代生物技术前沿)

ISBN 7-03-011493-0

Ⅰ. 计… Ⅱ. ①塞…②梅…③朱… Ⅲ. 分子生物学-计算方法 Ⅳ. Q7

中国版本图书馆 CIP 数据核字(2003)第 042257 号

策划编辑:周辉 王静/文案编辑:彭克里 吴慧涵/责任校对:朱光光
责任印制:刘士平 /封面设计:王浩

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

源海印刷有限责任公司 印刷

科学出版社发行 各地新华书店经销

*

2003年8月第 一 版 开本:787×1092 1/16
2003年8月第一次印刷 印张:15
印数:1—2 000 字数:339 000

定价:36.00 元

(如有印装质量问题,我社负责调换〈新欣〉)

译者序

计算生物学(包括生物信息学)是生物科学与技术在世纪之交迅猛发展的助推器。可以预见它将会发挥越来越广泛和重要的作用。如何在国内推动这个交叉学科的健康发展是一个紧要的问题。我们以为,好书所起的作用是不可低估的。在我们所接触过的书中,这一本书篇幅适中,内容全面,行文严谨,取材经典,是一本值得介绍的好书。特别要说明的是,原书的两位作者均是计算机科学家,读者也可从本书的行文风格体察得到。当他们多年前涉足计算生物学这条当时的“偏径”时,恐怕并不曾想到该学科今天能有这般红火。计算机和数学专业人员的参与推动了计算生物学近年在国外的迅速发展,我们希望国内也有越来越多的“外行”参与进来。

翻译是难以完美的,我们的原则首先是忠实原著,然后是力求通顺。术语的命名和索引的使用非常重要,我们推敲了所有术语的命名并重建了中文索引,希望该索引有助于读者的进一步阅读。需要说明的是,原文的索引并不全面,未收录术语的所有出处,我们并未对其作任何修改;其次,一些术语的译名与时下在其他书中采用的译法或用法不完全一致;最后,原书有一个勘误表,我们已根据勘误表校正了有关译文。对译文中可能存在的不当,我们事先请读者包涵。我们建议最好是阅读原文,而更好的办法则是:投身到计算生物学中来。

本书既可作为计算机专业、数学专业、生物医学工程专业高年级学生和研究生的教材,也可作为医学、生物学学生和工作者的参考书。本书基本上是纲领性质的,所呈示的内容是计算分子生物学的骨架。在弄懂了本书内容后,读者应能顺利阅读其他计算分子生物学和生物信息学文献。我们认为,尽管该领域发展很快,但大部分文献是关于应用的,因此从原理上弄通主要的脉络将大有裨益。

第一军医大学赵永忠博士翻译了部分初稿,国防科技大学计算机学院周斌博士、新加坡生物信息学研究所李国彬博士分别校对了第 2 章和第 3 章的译稿,在此特别致谢。同时感谢科学出版社和本书的责任编辑为本书的出版所付出的辛勤努力。

译校者　朱　浩

2002 年 12 月于新加坡生物信息学研究所

审校者　赵克森

2003 年 6 月于广州第一军医大学

前　言

生物学中有着至少500年也解决不完的有趣问题。

——Donnald E. Knuth

自从1953年DNA结构被揭示以来，分子生物学取得了巨大的进展。随着我们对生物大分子序列操纵能力的增强，科研工作已经产生并仍在产生大量的数据。处理来自全世界不同实验室所产生的大量数据，并使其可用于进一步的科研，产生了全新的在本质上具有多学科交叉性质的问题。生物科学家是这些数据的创造者和最终的用户。然而，由于数据的规模和复杂性，从创造到使用这些数据都需要多学科特别是数学和计算机科学的参与。这种要求产生了一个新的领域，通常叫做计算分子生物学。

广义地讲，计算分子生物学包括开发和使用数学与计算机技术，以帮助解决分子生物学中的问题。使用若干例子有助于说明。产生的所有信息都是以数据库的形式来存储的。国际上已存在好几个序列数据库，但生物科学家已认识到针对分子生物学的特殊需求，人们需要有新的数据库模型。例如，随着研究工作的进行，这些数据库应能记下我们对分子序列理解的变化，而目前的数据库尚不能满足这个要求。接下来，理解分子序列需要新的复杂的模式识别技术，而这种技术正是人工智能目前研究的课题。在数据库搜索过程中还遇到复杂的统计问题，这也需要新的专门工具。

而有一类问题最需要的是有效算法。简单地说，一个算法就是一个一步一步的过程，它能在限定时间内完成求解一个严格定义的问题。出于效率，算法在求解问题时不能花费“过长”的时间，即便问题相当的大。分子生物学中一个可以用算法求解的经典问题是序列比较：给定两个生物分子序列，我们要知道它们的相似性是多少。这样的问题每天要处理上千次，因此必须使用非常有效的算法。

本书的目的是介绍分子生物学中具有代表性的计算问题以及某些求解这些问题的有效算法。一些问题已相当清楚，某些算法也应用多年。另一些问题则困难些，尚未建立起满意的算法。对后一类问题，我们专注于阐释某些数学模型，这些模型可作为将来算法发展的基础。

读者应当知道，解决分子生物学问题的算法颇为奇特，它需要满足两类用户：分子生物学家和计算机科学家。前者要求算法切用，即要能求解带各种误差和不确定性的实际问题；后者则注重算法有效性的证明，证明它能有效地求解一个严格定义的问题，并愿意

为可证明性而牺牲切用性。我们试图在相互矛盾的二者之间寻求平衡,但时常倾向于后者,因为我们毕竟是计算机科学家。然而,我们希望本书对分子生物学家和计算科学家都有益。

这本书是个导论,这意味着我们的指导原则之一是尽可能展示我们认为简单的算法。对于书中的某些问题,存在着更有效亦更复杂的算法,有关这些算法的线索通常在每一章最后的文献提要中给出。尽管秉持简单性原则,本书中的一些算法和模型仍可能被认为是复杂的,这通常反映了对应问题所固有的复杂性,我们通常在标题前用“*”号将这些内容标出。这本书的导论性质还意味着对于某些课题,我们的介绍仅是个起点,这些课题完全可能需要一整本书的内容来讨论。

这本书的主要读者是数学和计算科学的学生。我们假定他们仅具有中学水平的分子生物学知识,因而提供专门一章简要介绍本书中用到的基本概念。对分子生物学不熟悉的读者应尽可能涉猎超出本书的内容,拓展他们的生物学知识。有关书籍在第一章的末尾给出。

我们希望本书在一定程度上也对生物科学学生有益,并假定这类读者在大学受过离散数学与算法方面的训练。为了帮助不熟悉这些内容的读者,我们安排了一章来简要介绍本书中用到的基本概念。

计算分子生物学进展迅速,更好的算法层出不穷,新的领域不断涌现。我们尽可能地覆盖主要领域,并认为大多数的材料具有永久价值。为满足想进一步研究的读者的需求,我们还提供了对若干信息源的索引(特别是在最后一章的文献提要,包括有用的网站)。当然,这些提示不是全面的。此外,我们不能保证所提供的网络 URL 仍然有效。尽管已测试过所有这些地址,但由于网络的动态特性,它们可能会有变动。

全 书 概 述

第 1 章介绍分子生物学的基本概念,主要包括蛋白质和核酸的基本结构和功能,分子遗传学的机制,研究生物基因组的主要实验室技术,以及现有序列数据库的概述。

第 2 章主要讲述串和图,它们是本书用到的两个最重要的数学对象。同时简短叙述了算法的基本概念,如何分析算法,NP-完全性的定义。

后续的章节主要针对分子生物学的特殊问题。第 3 章是序列比较,研究了基本的两序列比较并给出经典的动态程序设计算法。我们接着扩展了该算法,使之可用于该问题更一般的情形,其中一个是多序列比较,另一个是数据库搜索。我们亦讨论了有关的杂项问题。

第 4 章是片段组装。当 DNA 序列被打碎成小段时便引发了该问题,这些小段需要重新组装成原来的片段。这个技术被广泛应用于大规模测序项目,如人类基因组计划。我们揭示了各种因素可使该问题非常复杂。接着,介绍了几种简化的模型,并在最后部分给出了基于这些模型的算法和启发式。

第 5 章是 DNA 的物理标图问题,它可看做是更大尺度上的片段组装问题。由于片段更大,组装技术也完全不同,其目标是获得 DNA 分子上某些标记的位点。我们对主要

技术作了回顾并给出了模型,接着描述了连续1问题的算法,该抽象问题在物理标图中起重要作用。本章结尾部分讨论了一种物理标图的近似算法和启发式。

蛋白质和核酸随年代而进化,理解种系如何发生的一个重要工具是种系发生树,种系发生树亦能帮助理解蛋白质功能。第6章描述与种系发生树构造有关的一些数学问题,并给出某些用于构造特定类型种系发生树的算法。

计算生物学中一个新近出现的重要研究领域是基因重排。已经发现一些生物具有遗传差异,它们并非在序列水平有大的差异,而是大的序列块在各自DNA中的排序不同。已有一些有趣的数学模型用于研究这种差异。第7章专门论述这个问题。

理解分子的生物功能是计算生物学的核心问题。因为分子在三维空间里折叠,且因为其功能取决于这种折叠的方式,在过去几十年里科学家关心的一个主要议题是发现这些大分子的三维结构,特别是RNA和蛋白质的三维结构,这孕育出基于分子原始序列预测其结构的方法。第8章描述用于RNA结构预测的动态程序设计方法,概述了蛋白质结构预测的困难性,并介绍了该领域一个重要的近期进展,即蛋白质比配,它试图把蛋白质的序列和已知的结构作比较。

结束本书的第9章介绍一个令人兴奋的新领域:DNA计算。我们介绍了能使用DNA分子求解困难问题的生物学实验,并在理论上探讨了对另一个难题的求解。

我们已给出的一个约定是,标题带“*”的内容是作者认为较有难度的内容。在概念定义中,我们用到另一个约定,即第一次出现的全书均用到的术语以黑体表示,其他术语在定义中用斜体表示(译者注:译著中以楷体表示)。大多数算法首先以英语叙述,然后以伪代码描述(伪代码约定在第二章第二节列出)。在部分例子中,伪代码已相当具体,有助于感兴趣的读者将其实现。

较长的章节给出了小结。

习 题

每一章的结尾有习题。标记星号“*”的习题难度较大,可能需要超出本书的计算机知识,但一天之内应能求解。标记“**”的习题曾是研究课题,但现已解决,在文献提要中给出了解决该习题的原始文献。最后,标记“◇”的习题据作者所知到目前为止尚未解决。

书的最后我们给出部分习题的答案或提示。

错 误

尽管作者已尽其所能,错误仍在所难免。如果你发现错误,或有任何改进之建议,请不吝赐教。请将错误报告或其他评论寄至 bio@dcc.unicamp.br 或

J. Meidanis/J. C. Setubal

Instituto de Computacao, C.P.6176

UNICAMP

Campinas, SP 13083-970

Brazil
作者的私人电子邮箱是:meidanis@dcc.unicamp.br 或 setubal@dcc.unicamp.br。我们衷心感谢感兴趣的读者提出改正意见。查出的错误将在下面的网址公布:http://www.dcc.unicamp.br/~bio/ICMB.html。

致谢

本书是作者原先以葡萄牙文写于 1994 年并在巴西出版的第一部拙作的续篇。第一本书的出版要归功于两年一度的巴西计算机科学会议"Escola de Computacao",如果没有这个会议,我们就根本没有机会在此写序。

本书的诞生要感谢 Mike Sugarman、Bonnie Berger 和 Tom Leighton,他们给予我们很多鼓励和有帮助的建议。特别是 Bonnie,她甚至把她早期阶段的讲义拷贝给我们。

我们有幸得到来自 FAPESP 和 CNPq(巴西研究局)的资助,他们给我们多方面的帮助。FAPESP 基金用于"算法和组合学实验室"项目,提供了计算机设备;CNPq 基金是通过提供个人奖金以及通过 PROCOMB 和 TCPAC 资助研究访问的形式提供资助。

感谢我们学生对手稿的校对,特别是 Nalvo Franco de Almeida Jr. 和 Maria Emilia Machado Telles Walter。Nalvo 还制作了许多图表并给出很多建议。

我们和同事 Jorge Stolfi 进行了许多有益的讨论,他还给予至关重要的排版辅助。Fernando Reinach 和 Gilson Paulo Manfio 帮助我们编写了第 1 章。我们同 Jim Orlin,Martin Farach,Sampath Kannan 以及其他一些佚名的评阅人讨论了本书的目标和一般性问题,部分建议已写入书中。我们在 UNICAMP 计算研究所的同事给我们以鼓励并提供了令人上进的工作环境。

下列同行友好地寄给我们研究论文:Farid Alizadeh,Alberto Caprara,Martin Farach,David Greenberg,Dan Gusfield,Sridar Hannenhalli,Wen-Lian Hsu,Xiaoqiu Huang,Tao Jiang,John Kececioglu,Lukas Knecht,Rick Lathrop,Gene Myers,Alejandro Schäffer,Ron Shamir,Martin Vingron(他还寄来了讲义),Todd Wareham 和 Tandy Warnow。本书的一些部分很大程度上基于上述的部分论文。

非常感谢 Erik Brisson,Eileen Sullivan,Bruce Dale,Carlos Eduardo Ferreira 和 Thomas Roos,他(她)们以不同的方式提供了帮助。

J.C.S 感谢他的妻子 Silvia(a.k.a Teca)和孩子 Claudia,Tomás 和 Caio,没有他们的支持本书可能无法完成。

作者以 Leslie Lamport 的 LATEXA2 系统排版本书。

前言开头引用的语句出自 Don Knuth 在 1993 年 12 月 7 日同 Computer Literacy Bookshops,Inc. 的谈话。

João Carlos Setubal
João Meidanis

目　录

译者序 …………………………………………………………………………… (i)
前言 ……………………………………………………………………………… (iii)
全书概述 …………………………………………………………………………… (iv)
习题 ………………………………………………………………………………… (v)
错误 ………………………………………………………………………………… (v)
致谢 ………………………………………………………………………………… (vi)
第 1 章　分子生物学的基本概念 ……………………………………………… (1)
1.1　生命……………………………………………………………………………… (1)
1.2　蛋白质…………………………………………………………………………… (2)
1.3　核酸……………………………………………………………………………… (4)
1.3.1　DNA ………………………………………………………………………… (4)
1.3.2　RNA ………………………………………………………………………… (6)
1.4　分子遗传学机制………………………………………………………………… (6)
1.4.1　基因和遗传密码 …………………………………………………………… (7)
1.4.2　转录、翻译和蛋白质合成 ………………………………………………… (8)
1.4.3　无用 DNA 和读框 ………………………………………………………… (9)
1.4.4　染色体 ……………………………………………………………………… (10)
1.4.5　基因组类似计算机程序? …………………………………………………… (11)
1.5　基因组是如何被研究的………………………………………………………… (11)
1.5.1　图谱和序列 ………………………………………………………………… (11)
1.5.2　特殊技术…………………………………………………………………… (13)
1.6　人类基因组计划………………………………………………………………… (16)
1.7　序列数据库……………………………………………………………………… (17)
习题 ………………………………………………………………………………… (23)
文献提要 …………………………………………………………………………… (23)
第 2 章　串、图和算法…………………………………………………………… (24)
2.1　串………………………………………………………………………………… (24)
2.2　图………………………………………………………………………………… (25)
2.3　算法……………………………………………………………………………… (28)
习题 ………………………………………………………………………………… (32)
文献提要 …………………………………………………………………………… (33)
第 3 章　序列比较与数据库搜索 ……………………………………………… (34)
3.1　生物学背景……………………………………………………………………… (34)

3.2 比较两个序列…………(35)
3.2.1 全局比较—基本算法 …………(35)
3.2.2 局部比较…………(40)
3.2.3 半全局比较…………(41)
3.3 基本算法的扩展…………(42)
3.3.1 空间节省…………(42)
3.3.2 一般空隙罚分函数 …………(45)
3.3.3 仿射空隙罚分函数 …………(47)
3.3.4 比较相似序列 …………(49)
3.4 比较多个序列…………(52)
3.4.1 SP度量 …………(52)
3.4.2 星比对 …………(58)
3.4.3 树比对 …………(59)
3.5 数据库搜索…………(60)
3.5.1 PAM矩阵 …………(60)
3.5.2 BLAST …………(63)
3.5.3 FAST …………(65)
3.6 其他问题…………(67)
*3.6.1 相似性与距离 …………(67)
3.6.2 序列比较中的参数选择 …………(72)
3.6.3 串匹配和确切序列比较 …………(74)
小结 …………(75)
习题 …………(76)
文献提要 …………(77)
第4章 DNA片段组装 …………(80)
4.1 生物学背景…………(80)
4.1.1 理想情形…………(81)
4.1.2 复杂情形…………(81)
4.1.3 DNA测序的其他方法 …………(86)
4.2 模型…………(87)
4.2.1 最短公共超串 …………(87)
4.2.2 重构 …………(88)
4.2.3 多连叠 …………(89)
*4.3 算法 …………(91)
4.3.1 交叠的表示…………(91)
4.3.2 通路产生超串 …………(91)
4.3.3 作为通路的最短超串 …………(93)
4.3.4 贪婪算法…………(95)

4.3.5 无环子图 …… (96)
4.4 启发式 …… (100)
4.4.1 发现交叠 …… (102)
4.4.2 排序片段 …… (103)
4.4.3 比对与表决 …… (104)
小结 …… (106)
习题 …… (106)
文献提要 …… (107)
第 5 章 DNA 物理作图 …… (109)
5.1 生物学背景 …… (109)
5.1.1 限制位点作图 …… (110)
5.1.2 杂交作图 …… (111)
5.2 模型 …… (112)
5.2.1 限制位点模型 …… (112)
5.2.2 区间图模型 …… (113)
5.2.3 连续 1 特性 …… (115)
5.2.4 算法启示 …… (115)
5.3 一个 C1P 问题的算法 …… (116)
5.4 带错杂交作图的一种近似 …… (122)
5.4.1 一个图模型 …… (123)
5.4.2 一个保证 …… (124)
5.4.3 实际计算 …… (126)
5.5 杂交作图的启发式 …… (128)
5.5.1 筛选嵌合克隆 …… (128)
5.5.2 获得一个好的探针次序 …… (129)
小结 …… (130)
习题 …… (130)
文献提要 …… (132)
第 6 章 种系发生树 …… (133)
6.1 性状状态和完全种系发生问题 …… (135)
6.2 二值性状状态 …… (138)
6.3 两个性状 …… (141)
6.4 种系树的简约性和相容性 …… (144)
6.5 距离矩阵算法 …… (146)
6.5.1 重构可加树 …… (146)
* 6.5.2 重构超度量树 …… (149)
6.6 种系树之间的一致 …… (155)
小结 …… (158)

习题……………………………………………………………………………… (159)
文献提要…………………………………………………………………………… (160)
第7章 基因组重排……………………………………………………………… (163)
7.1 生物学背景 ……………………………………………………………… (163)
7.2 有向块 …………………………………………………………………… (165)
7.2.1 若干定义 ……………………………………………………………… (166)
7.2.2 断点 …………………………………………………………………… (167)
7.2.3 现实与期望图式 ……………………………………………………… (168)
7.2.4 交叉图 ………………………………………………………………… (172)
7.2.5 坏组分 ………………………………………………………………… (175)
7.2.6 算法 …………………………………………………………………… (176)
7.3 无向块 …………………………………………………………………… (178)
7.3.1 条……………………………………………………………………… (179)
7.3.2 算法 …………………………………………………………………… (181)
小结……………………………………………………………………………… (182)
习题……………………………………………………………………………… (182)
文献提要…………………………………………………………………………… (183)
第8章 分子结构预测…………………………………………………………… (185)
8.1 RNA二级结构预测……………………………………………………… (185)
8.2 蛋白质折叠问题 ………………………………………………………… (190)
8.3 蛋白质比配 ……………………………………………………………… (191)
小结……………………………………………………………………………… (195)
习题……………………………………………………………………………… (195)
文献提要…………………………………………………………………………… (195)
第9章 结语:DNA计算 ………………………………………………………… (197)
9.1 哈密顿通路问题 ………………………………………………………… (197)
9.2 可满足性 ………………………………………………………………… (199)
9.3 问题与展望 ……………………………………………………………… (202)
习题……………………………………………………………………………… (202)
文献提要…………………………………………………………………………… (203)
习题选解………………………………………………………………………… (204)
参考文献………………………………………………………………………… (209)
索引……………………………………………………………………………… (220)

第1章　分子生物学的基本概念

本章介绍分子生物学的基本概念，旨在为读者提供足够的信息，以便他们能从容应对本书中涉及的生物学背景以及一般的计算分子生物学文献。受过严格科学训练的读者应清楚在分子生物学中没有百分之百绝对的东西，每一条规则都有例外。对于一般的规则，我们已尽力指出某些主要的例外，而在其他情形下则常常省略不提。所以读者不宜将本章视为分子生物学的教科书。

1.1　生命

自然界包括生命和非生命物质。相对于非生命物质而言，生命物质具有移动、繁殖、 1[①]
生长和摄食等能够主动参与环境的特性。20世纪的研究已表明这两类物质都是由同样的原子组成并遵循同样的物理和化学规律。那么，区别何在？在长久的人类历史中，人们认为生命体被赋予了某些特殊的物质，使得它们具备主动的特性，即它们被这种物质“激活”了。可是，这类物质却从未被发现。相反，我们现在知道生命体的行为缘于它们体内一系列复杂的化学反应。这些反应从不停止，且常常是一个反应的产物即刻被另一个反应利用，使系统往复不止。活着的生物持续不断地与周围环境进行物质和能量交换；反
之，任何与环境达到平衡的东西都被认为是死亡的(某些生物形态则是例外，如种子、病 2
毒，它们可长时间没有活性，但并不死亡)。

现代科学研究表明生命起源于35亿年前，地质时间上稍晚于(按地质术语)地球本身的形成。最初的生命形式非常简单，但在几十亿年的称之为进化的主动过程的作用下，生命发生了演绎并产生了多样性，因此今天我们可以同时看到非常复杂和非常简单的生命体。

复杂和简单的生物有着相似的分子化学或生物化学。生物化学中的主要角色是被称为**蛋白质**(protein)和**核酸**(nucleic acid)的分子。粗略地说，蛋白质决定一个生物是什么和做什么(著名的科学家Russell Doolittle曾写过“我们是我们的蛋白质”)，核酸则负责编码产生蛋白质所必要的信息，并把这种信息传给后代。

分子生物学的研究基本上致力于理解蛋白质和核酸的结构与功能，这些分子因而成

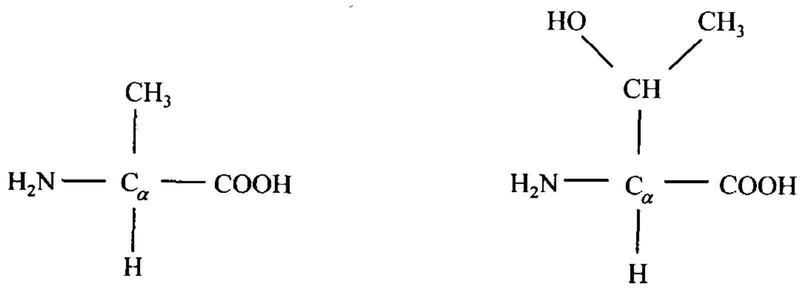
① 为了方便读者查阅，我们在正文切口空白处加上原书的原始页码。

为本书的研究对象，下面我们将简要介绍当前有关蛋白质和核酸的基本知识。

1.2　蛋白质

我们身体的大部分物质是各种各样的蛋白质。蛋白质包括很多种，结构蛋白是组织的构成单元，酶是化学反应的催化剂。催化剂是指能够加速化学反应的物质。如果没有催化剂，大多数生物化学反应进行得很慢甚至不能开始。酶可呈数量级地加速这种过程，使生命成为可能。酶非常特别，通常一种酶仅催化一类生物化学反应。考虑到为维持生命需要有大量的生化反应，我们就需要大量的酶。蛋白质的其他功能还包括氧气运输和抗体防御等。究竟什么是蛋白质？它们如何组成？它们如何发挥功能？这一部分将简要回答这些问题。

蛋白质是由一类称为**氨基酸**(amino acid)的简单分子组成的。图 1.1 是氨基酸的例子，每一个氨基酸有一个中心碳原子，谓之 α 碳原子或 C_α。C_α 连接 1 个氢原子(H)、1 个氨基($-NH_2$)、1 个羧基($-COOH$)和 1 个侧链。正是侧链决定了氨基酸间的差异。侧链可以是简单的氢原子(如甘氨酸)，也可以是复杂的两个碳环(如色氨酸)。我们在自然界只发现了 20 种不同的氨基酸，表 1.1 列出了这些氨基酸，这是蛋白质中最常见的 20 种，
3 另外还有几种非标准的氨基酸。



丙氨酸　　苏氨酸

图 1.1　氨基酸的例子

在蛋白质中，氨基酸通过肽键相连。因此，蛋白质是多肽链。在肽键中，属于氨基酸 A_i 的羧基的 C 原子与氨基酸 A_{i+1} 的 N 原子连接。在该键形成中，因羧基的氢原子和氧原子与氨基的氢原子结合而脱去 1 个水分子。因此我们在肽键内部所发现的仅仅是一个**残基**(residue)。由此我们说蛋白质有 100 个残基而不是说有 100 个氨基酸。典型的蛋白质含大约 300 个残基，但也有少至 100 个残基和多达 5 000 个残基的氨基酸。

表 1.1　蛋白质中发现的 20 种常见氨基酸

	单字母缩写	三字母缩写	名　称
1	A	Ala	丙氨酸(alanine)
2	C	Cys	半胱氨酸(cysteine)
3	D	Asp	天冬氨酸(aspartic acid)
4	E	Glu	谷氨酸(glutamic acid)

续表

	单字母缩写	三字母缩写	名称
5	F	Phe	苯丙氨酸(phenylalanine)
6	G	Gly	甘氨酸(glycine)
7	H	His	组氨酸(histidine)
8	I	Ile	异亮氨酸(isoleucine)
9	K	Lys	赖氨酸(lysine)
10	L	Leu	亮氨酸(leucine)
11	M	Met	甲硫氨酸(methionine)
12	N	Asn	天冬酰胺(asparagine)
13	P	Pro	脯氨酸(proline)
14	Q	Gln	谷氨酰胺(glutamine)
15	R	Arg	精氨酸(arginine)
16	S	Ser	丝氨酸(serine)
17	T	Thr	苏氨酸(threonine)
18	V	Val	缬氨酸(valine)
19	W	Trp	色氨酸(tryptophan)
20	Y	Tyr	酪氨酸(tyrosine)

4

肽键使得每个蛋白质都有一个骨架(backbone),它是重复的基本单元—N—C_α—(CO)—,每个 C_α 有一个侧链,图 1.2 是一个多肽链的图示。因为在骨架的一端是一个氨基,另一端是一个羧基,我们因此可以区别多肽链的两端并给它定一个方向,习惯上多肽始于氨基(N 端),止于羧基(C 端)。

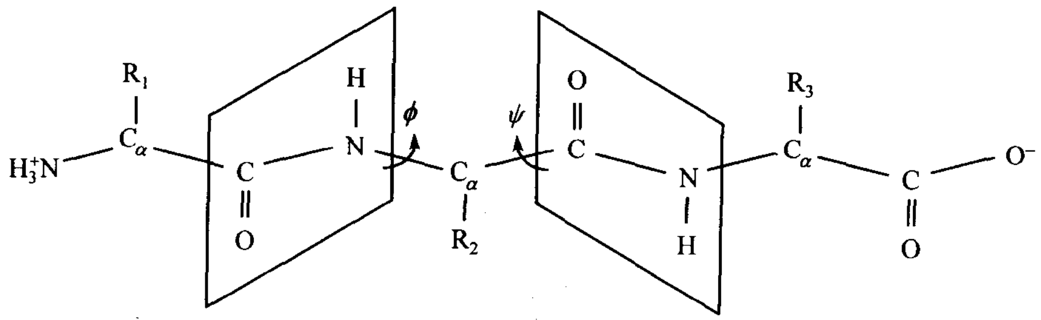


图 1.2　一个多肽链,R_i 侧链表示氨基酸,每个四边形中的原子在同一平面上,能够根据角度 ϕ 和 ψ 旋转

蛋白质并不仅仅是氨基酸残基的线性序列,这种序列称为**一级结构**(primary structure)。蛋白质实际上在三维空间中折叠,形成**二级**(secondary)、**三级**(tertiary)和**四级**(quaternary)结构。蛋白质的二级结构是通过骨架原子间的相互作用形成的,并导致"局部"结构,如螺旋(helix)等。三级结构是二级结构在更大的范围内堆积的结果。更高层次的堆积,即一组不

同亚基(译者注:原书此处是"protein")的堆积,形成四级结构。图 1.3 显示了这些结构。

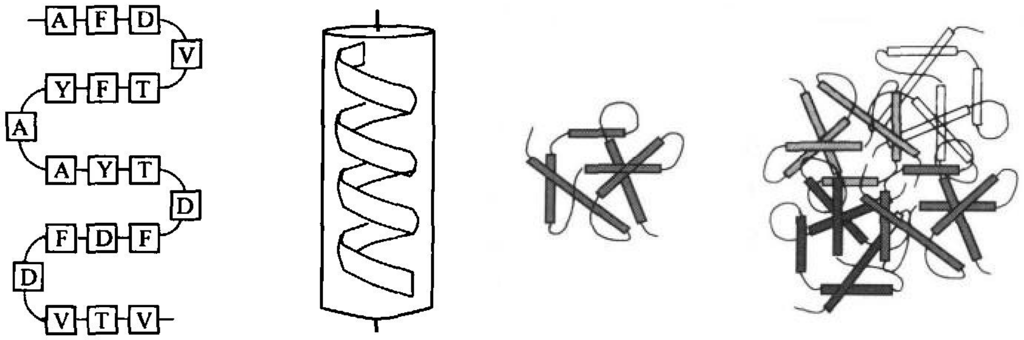


图 1.3　蛋白质的一级、二级、三级、四级结构(取自参考文献[28]中的图)
译者注:第二个图显示了一个左手螺旋,而实际上蛋白质中几乎都是右手螺旋

蛋白质之所以能够在三维空间中折叠是因为 C_α 原子和 N 原子之间的肽键平面可以旋转,C_α 原子和其他 C 原子之间的平面也可以旋转,如图 1.2 所示,这两类旋转的角度分别表示为 ϕ 和 ψ。侧链也可以运动,但相对于骨架旋转而言属于次级运动。因此只要我们规定了蛋白质中所有 ϕ-ψ 对的值,便可确切知道蛋白质的折叠。确定蛋白质的折叠或三维结构是分子生物学的一个重要领域,原因有三点。首先,蛋白质的三维结构与其功能相关;其次,蛋白质由 20 种氨基酸组成,这使得三维结构非常复杂,缺乏对称性;第三,目前没有简单精确的确定三维结构的方法。这些因素促成了第 8 章内容的形成,我们在该部分讨论分子结构的预测方法,这些方法试图通过分子的原始序列预测其结构。

蛋白质的三维形状能以下述方式决定其功能。折叠的蛋白质具有不规则的形状,这
5 意味着它具有各种隐窝和膨凸,这些形状使得蛋白质能够与其他特定分子形成紧密的接触或**结合**(bind)。哪些分子可以结合取决于蛋白质的形状。例如,蛋白质的形状可以容许自身几个拷贝的结合,形成一个像发丝的结构;或其结构可使得分子 A 和 B 与其结合并交换原子,A 和 B 之间发生了化学反应,而蛋白质起到了催化剂的作用。

我们是如何获得蛋白质的?蛋白质是在一个称为核糖体的细胞结构中合成的。在核糖体中,根据**信使 RNA**(messenger ribonucleic acid,mRNA)分子所包含的信息,蛋白质的氨基酸一个接一个地装配起来。为解释这是如何发生的,我们需要说明什么是核酸。

1.3　核酸

生物体包含两类核酸:**核糖核酸**(ribonucleic acid),简写为 RNA;**脱氧核糖核酸**(deoxyribonucleic acid),简写为 DNA。我们首先描述 DNA。

1.3.1　DNA

类似于蛋白质,DNA 分子也是由小分子组成的链。实际上 DNA 是双链,但我们首先认识一下 DN A 的**单链**(strand),它是由重复的基本单元组成的骨架。这种基本单元由一个称做 2′-脱氧核糖的糖分子和一个磷酸残基组成。糖分子含有 5 个碳原子,标记为 1′→

5′(图 1.4)。形成骨架的键是在一个单元的3′碳原子和下一个单元的5′碳原子之间。因此,DNA 分子具有**方向性**(orientation),一般从5′开始到3′结束。当我们在技术文献、书籍和序列数据库看到一个单链 DNA 分子时,若无特别说明,均是按5′→3′方向书写的。 6

与骨架中与1′碳原子相连的分子为**碱基**(base)。在 DNA 分子中有4种碱基,分别是:腺嘌呤(adenine,A)、鸟嘌呤(guanine,G)、胞嘧啶(cytosine,C)和胸腺嘧啶(thymine,T)。图 1.5 显示了每种碱基的分子结构,图 1.6 为单链 DNA 分子的示意图。碱基 A 和 G 是嘌呤,而碱基 C 和 T 属于嘧啶。DNA 分子的

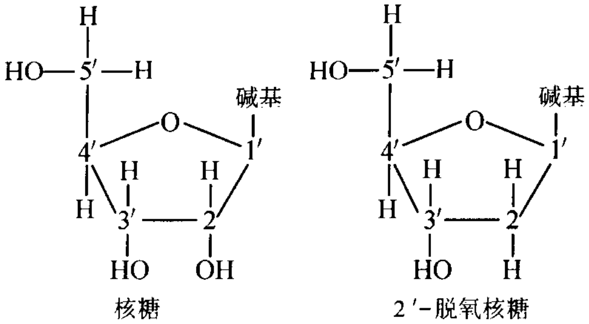


图 1.4　核酸中的糖,符号1′→5′表示碳原子。两种糖的惟一差别是2′碳原子中的氧,RNA 中是核糖,DNA 中是脱氧核糖

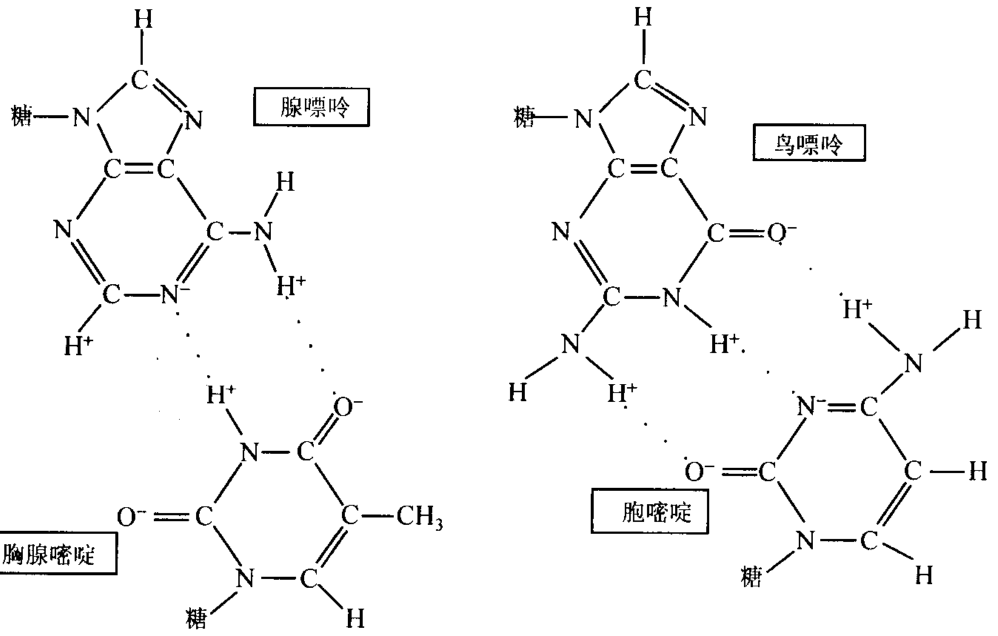


图 1.5　DNA 中氮化的碱基。注意腺嘌呤和胸腺嘧啶、鸟嘌呤和胞嘧啶之间所形成的键,图中用点画线表示

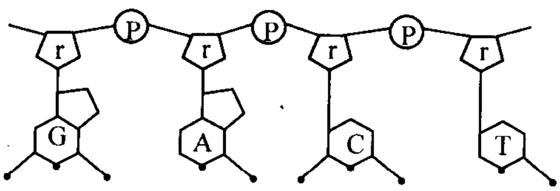


图 1.6　一个 DNA 链分子结构的图示

基本单元由糖、磷酸和碱基组成,该基本单元谓之**核苷酸**(nucleotide)。虽然碱基和核苷酸不同,但我们可以称 DNA 分子有 200 个碱基或由 200 个核苷酸组成。仅含有几个或几十个核苷酸的 DNA 分子称为寡核苷酸。自然界中 DNA 分子非常长,长度远超过蛋白质。在人的细胞中,DNA 分子由上亿个核苷酸组成。

前已述及,DNA 分子是双链结构。两条链缠绕在一起形成双螺旋,此著名的双螺旋(double helix)结构是由 James Watson 和 Francis Crick 在 1953 年发现的。两条链结合的

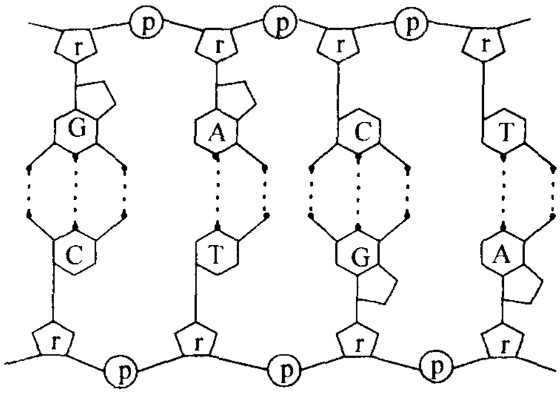


图 1.7　双链 DNA 分子结构的图示

机制是一条链的碱基与另一条链的碱基配对，碱基 A 始终与碱基 T 配对，碱基 C 始终与碱基 G 配对，如图 1.5 和图 1.7 所示。碱基 A 和 T 称做彼此**互补**(complement)，或**互补碱基对**(complementary base)；类似地，C 和 G 是互补的。这种碱基对亦称为沃森-克里克碱基对(Watson-Crick base pair)。碱基对通常用于表示 DNA 长度，缩写为 bp。所以我们可以说一段 DNA 的长度是100 000bp，或 100kb。

在本书中我们将视 DNA 为字母串，每一个字母代表一个碱基。图 1.8 是 DNA 的"串表示"，将一串字母置于另一串字母之上来表示双链 DNA。请注意碱基配对，尽管两条链是相连的，但每一链有自己的方向，

5′…TACTGAA…3′
3′…ATGACTT…5′

图 1.8　双链 DNA 表达为字母串

7 两条链方向相反，一条链的 3′ 端对应另一条链的 5′端。这种特性有时称做反向平行(antiparallel)，它的一个重要结果是可从一条链推断出另一条链，这种操作称为反向互补，或**逆补**(reverse complementation)。例如，给定标准方向的一条链 s = AGACGT，我们可用下面的操作获得其逆补。首先，反向 s，得到 s' = TGCAGA，然后做互补碱基替换，得到 $\bar{s}$ = ACGTCT(注意我们用上短杠标记 s 的逆补链)。正是这种精密的机制使得 DNA 在细胞中得以复制，使得有机体从一个细胞长成数以十亿计的细胞，每个细胞均含有起源细胞的 DNA 分子。

在没有细胞核的细胞中，DNA 自由漂浮在细胞内。在高等生物体，DNA 分子在细胞
8 核或细胞器中，后者包括动物和植物的线粒体以及植物的叶绿体。

1.3.2　RNA

RNA 分子与 DNA 分子非常相似，但有下列组成和结构上的不同：

- RNA 中，核糖取代了 DNA 分子中的 2′-脱氧核糖(如图 1.4 所示)。
- RNA 中，胸腺嘧啶 T 被尿嘧啶 U 取代，U 与 T 一样能够与 A 配对。
- RNA 并不形成双螺旋。有时我们可看到 RNA-DNA 杂交双螺旋。此外，RNA 能够通过碱基互补与同一分子的其他部分结合。RNA 的三维结构远较 DNA 复杂。

DNA 与 RNA 的另一个不同是，DNA 本质上仅有一个功能，即编码信息，而我们接下
9 来会看到细胞中有不同的 RNA，各自行使不同的功能。

1.4　分子遗传学机制

DNA 分子的重要性在于，机体中合成 RNA 和蛋白质的信息编码于 DNA 分子中。基于此，DNA 有时被称为"生命蓝图"。在这一部分，我们将描述这种编码以及蛋白质如

何根据 DNA 中的信息被构造(这一过程称为蛋白质合成)。我们还会看到 DNA 中的信息,即遗传信息,如何从亲代传至子代。

1.4.1　基因和遗传密码

有机体的每一个细胞都有几个非常长的 DNA 分子,每一个这样的分子称为**染色体**(chromosome)。我们在后面的部分会讨论染色体,此时让我们仅关注于 DNA 分子的信息编码方式。首先要知道的是 DNA 中仅有一部分连续的片段编码构建蛋白质的信息,而其他部分并不编码蛋白质信息。第二点是每一种不同的蛋白质仅对应一段 DNA 序列,该段序列称为**基因**(gene)。因为某些基因编码 RNA 分子,因此更正确地说基因是编码蛋白质或 RNA 的连续的 DNA 序列。基因长度变化很大,人类的基因可为 10 000bp。特殊的细胞机制能够准确识别基因的起点和终点。

如前所述,蛋白质是氨基酸链。因此,确定蛋白质仅需确定其所含的每一个氨基酸,这正是基因所要做的,它用三联核苷酸编码每一个氨基酸。每个三联核苷酸称为**密码子**(codon)。三联核苷酸与氨基酸之间的对应关系称为**遗传密码**(genetic code),见表 1.2。在该表中你会发现三联核苷酸是用 RNA 碱基而非 DNA 碱基表示,其原因是,RNA 分子提供了 DNA 和蛋白质之间的关联。在详细叙述这种关联之前,我们先详尽研究一下遗传密码。

表 1.2　氨基酸的遗传密码

第一位置	第二位置				第三位置
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
	Gly	Glu	Ala	Val	A
	Gly	Asp	Ala	Val	C
	Gly	Asp	Ala	Val	U
A	Arg	Lys	Thr	Met	G
	Arg	Lys	Thr	Ile	A
	Ser	Asn	Thr	Ile	C
	Ser	Asn	Thr	Ile	U
C	Arg	Gln	Pro	Leu	G
	Arg	Gln	Pro	Leu	A
	Arg	His	Pro	Leu	C
	Arg	His	Pro	Leu	U
U	Trp	STOP	Ser	Leu	G
	STOP	STOP	Ser	Leu	A
	Cys	Tyr	Ser	Phe	C
	Cys	Tyr	Ser	Phe	U

注意存在 64 种可能的三联密码子,但氨基酸仅有 20 种,这导致存在不同的三联密码子对应同一氨基酸。例如,AAG 和 AAA 都编码赖氨酸。另外,有 3 个密码子并不编码任何氨基酸而是用于表示基因的终止,这 3 个特殊的终止密码在表 1.2 中用 STOP 表示。

10 最后，我们要注意以上所述的是绝大多数生物应用的遗传密码，有些生物有少量的改动。

1.4.2　转录、翻译和蛋白质合成

现在我们描述 DNA 信息是如何指导蛋白质合成的。一个识别基因或基因簇起始的机制是*启动子*(promoter)。启动子是基因前面的一段 DNA 序列，指征位于其前面的基因。密码子 AUG(编码甲硫氨酸)则是基因开始的信号。识别出基因的起始点后，基因到 RNA 的拷贝就开始了，合成的 RNA 为信使 RNA(messager RNA)，简写为 mRNA，其序列与 DNA 中的一条链相同，但 U 代替了 T，该过程称为**转录**(transcription)。mRNA 被用在称为核糖体的细胞结构内合成蛋白质。

因 RNA 是单链而 DNA 是双链的，mRNA 在序列上与一条 DNA 链相同(仅是 U 取代了 T)，与另一条链互补。DNA 中，与 mRNA 序列一致的链称为*反义链*(antisense strand)或*编码链*(coding strand)，而另一链称为*有意义链*(sense strand)、*反密码链*(anticoding strand)或*模板链*(template strand)。实际上被转录的是模板链，因为 mRNA 是由与该链互补的核糖核酸组成的，合成过程是从 5′端向 3′端，因此模板链是从 3′端向 5′端阅读的。注意并非某一条链永远是模板链，例如，基因 *A* 的模板链是某一条链，而基因 *B* 的模板链则是另一条链。因为有启动子，对于一给定的基因，细胞能够识别对应的模板链。即便启动子的互补序列出现在另一链上，这个逆补也并不是启动子，不能被识别。这一事实的一个重要后果是，同一染色体上的基因相互之间具有方向性，给定两个基因，如果它们在同一链上则为同向，否则为反向，
11 这是第 7 章的基本点。最后我们注意术语*上游*(upstream)和*下游*(downstream)，它们用来指征 DNA 的相对于编码链方向的位置，启动子位于基因的上游。

上文所述的转录在*原核生物*(prokaryote)中是正确的。在这些生物中，由于缺乏核膜，DNA 是自由悬浮于细胞中的。原核生物的代表是细菌和蓝藻。所有其他生物归类为*真核生物*(eukaryote)，它们有细胞核，DNA 位于细胞核中，其转录模式更为复杂。许多真核细胞基因具有不同的组成部分，即*内含子*(intron)和*外显子*(exon)。转录后，内含子必须从 mRNA 中切除，这意味着内含子不参与蛋白质的合成。编码牛心房肽的基因是一个典型例子，该基因有1082bp，有 3 个外显子，分别位于1～120，219～545，1071～1082，有 2 个内含子，分别位于 121～218，546～1070。所以，编码碱基仅有 459 个，而对应的蛋白质有 153 个氨基酸。在内含子被剪切后，离开细胞核的剪短的 mRNA 仅含有外显子与起始和终止的调控序列。

因为有内含子和外显子，我们用不同的名字命名染色体上的全基因和剪切后仅含外显子的基因，前者称*基因组* DNA(genomic DNA)，后者称*互补* DNA(complementary DNA)或 cDNA。科学家可在不知道基因组 DNA 的情况下合成 cDNA。他们首先捕获细胞核外的 mRNA，然后使用一个称为*反转录*(reverse transcription)的过程，以 mRNA 为模板合成 cDNA。因为 mRNA 仅含外显子，合成的 DNA 便也仅含外显子。因此，人们可以不经染色体而获得 cDNA。转录和反转录都是需要酶催化的复杂过程，转录酶和反转录酶是细胞中分别催化这些过程的酶。另外一种现象是*选择性剪切*(alternative splicing)，即通过选择不同的内含子和外显子组合，相同的基因组 DNA 能够产生两个或更多的 mRNA，它们通常产生不同的蛋白质。

现在让我们重新回到 mRNA 和蛋白质合成。在这个过程里另外两种 RNA 分子起到重要的作用。如前所述,蛋白质合成是在细胞内被称为核糖体的细胞器中进行的。核糖体由蛋白质和被称为核糖体 RNA(ribosomal RNA,rRNA)的大分子组成,它的功能类似于工厂的装配线,输入 mRNA 和另外一种叫做转移 RNA(transfer RNA,tRNA)的分子,输出蛋白质。

实际上,遗传密码的翻译是由 tRNA 实现的,这个过程称为翻译(translation),它连接 12
密码子和其所编码的氨基酸。每一个 tRNA 分子在一端有一个对特定密码子有高亲和力的构象(conformation),在另一端有一个容易与该密码子对应的氨基酸结合的构象。当 mRNA 穿过核糖体内部时,tRNA 匹配当前的密码子,即当前位于核糖体内部的 mRNA 密码子,与它结合,并带进对应的氨基酸(细胞中总是悬浮着大量的氨基酸)。这些分子此时所处的空间位置使它们可以完成下列操作,即随着 tRNA 与密码子结合,新结合的氨基酸紧靠先前已形成的氨基酸链,一个合适的酶则催化该氨基酸加入肽链,然后释放tRNA。蛋白质按这种一个氨基酸接一个氨基酸的方式合成起来。当出现终止密码子时,没有 tRNA 与之对应,合成便终止,mRNA 被释放,并被降解成核糖核苷酸,降解物可循环用于其他 RNA 合成。

有人或许会认为,tRNA 的数量应与密码子数量相同,实际并不是这样。实际 tRNA 数量随物种而变化。例如,大肠杆菌(*E.coli*)大约有 40 种 tRNA。有些密码子对应的 tRNA 并不出现,有些 tRNA 可结合多个密码子。

图 1.9 总结了刚才我们描述的过程,我们对细胞内遗传信息流动的观点通常用中心法则(central dogma)来说明。

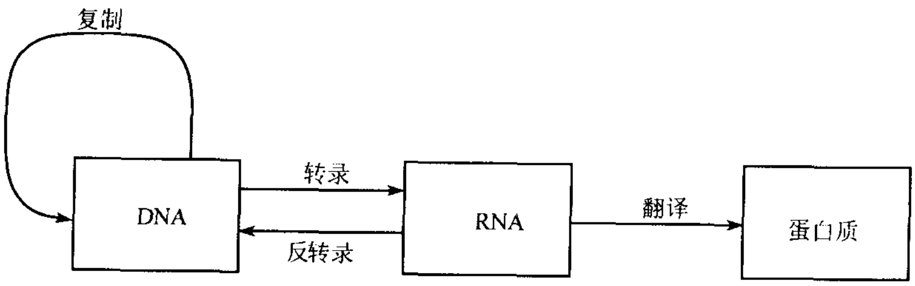


图 1.9　细胞内的遗传信息流动:分子生物学的中心法则

1.4.3　无用 DNA 和读框

在这一部分中,我们将对上一段的描述提供一些额外细节。前已述及,基因是染色体上的连续区域,但并没有占据整个染色体。每一个基因,或一组相关基因,被调节区域包 13
围。调节区域控制基因的转录和其他有关过程。但亦存在无已知功能的基因间区域,称为无用 DNA(junk DNA),因为它们的出现看起来并无特定的理由。而且,这个区域可积累许多碱基突变,因为位于这些部位的不影响基因和调控序列的突变常常不是致死性的,因而能够传给后代。然而,最近的研究显示无用 DNA 含有比先前所认为的更多的信息。无用 DNA 的量随物种而变化。原核生物几乎没有,其染色体基本上为基因所覆盖。相

比之下，真核生物则有大量的无用 DNA，如人大约有 90% 的染色体 DNA 是无用 DNA。

理解读框(reading frame)对理解转录过程非常重要。读框是指把 DNA 或 RNA 序列划分成密码子的 3 种可能方式之一。例如，考虑序列 TAATCGAATGGGC，一种读框产生的密码子排列是 TAA、TCG、ATT、GGG，剩下 1 个 C；另一种读框是忽略第 1 个 T，获得密码子 AAT、CGA、ATG、GGC；还有一种读框将产生 ATC、GAA、TGG 密码子，余下开始的 2 个碱基(TA)和最后的 2 个碱基(GC)。

需注意的是 3 种读框分别起始自给定序列的 1、2、3 位置。如果我们考虑从位置 4 开始，获得的密码子是起自碱基 1 所获得读框的子集，即是起自不同位置的同一读框。通常，如果分别起始自 i 和 j，$j-i$ 是 3 的倍数，那么实际上我们使用的是同一读框。

有时我们讨论序列中的 6 个而不是 3 个不同的读框，在这种情况下，我们考虑的是两条 DNA 序列的情况。我们在一条链上获得 3 个读框，在另一条链上有另外 3 个，总共有 6 个读框。当我们获得新测序的 DNA 并将它与蛋白质数据库相比较时经常要做这样的工作。我们需将 DNA 序列翻译成蛋白质序列，但共有 6 种方式，每一种具有不同的读框。在序列的末端丢失 1 或 2 个碱基并不重要，这些序列足够长，即便缺少几个氨基酸也能产生有意义的序列比较。

DNA 序列中的一个可读框(open reading frame，ORF)是指该序列的一个连续段，由起始密码子开始，含有整数个的密码子(长度是 3 的倍数)，且无终止密码子。起始密码子前面的调控序列也用于分析 ORF。

1.4.4 染色体

在这一段我们简要叙述染色体水平的遗传信息传递。首先我们需知道，一个细胞内的全套染色体称为**基因组**(genome)。基因组中染色体的数量是物种的一个特征。例如，
14 人的每一个细胞中有 46 条染色体，而小鼠有 40 条。表 1.3 列出了几个物种的染色体个数和基因组碱基对的数量。

表 1.3 某些物种的基因组大小，除了我们人类本身，表中所列生物在分子生物学和遗传学研究中很重要

物　种	染色体数(二倍体)	基因组大小/bp
噬菌体 λ	1	5×10^4
大肠杆菌	1	5×10^6
酵母	32	1×10^7
线虫	12	1×10^8
果蝇	8	2×10^8
人	46	3×10^9

原核生物通常只有 1 条染色体，有时是环型的。而真核生物染色体成对出现(含有成对染色体的细胞称为二倍体细胞(diploid)。例如，人有 23 对染色体，每对中的染色体分别来自父母。配对的染色体称为同源染色体(homologous)，配对染色体的基因相互对应。某些来自父母双方的基因完全一样，如血红蛋白基因，该蛋白的主要功能是在血液中携带氧气。其他来自父母双方的基因可有不同的形式，称为等位基因(allele)。典型的例子是人的血型基因，有

A、*B* 和 *O* 3 种形式。众所周知，如果一个人从母亲获得 *A* 等位基因而从父亲获得 *B* 等位基因，则他的血型应为 *AB* 型。

仅含有单套染色体的细胞称为*单倍体细胞*(haploid)，这些主要是在有性繁殖中产生的生殖细胞。当来自母亲的单倍体细胞与来自父亲的单倍体细胞相融合时，形成受精卵，重新回到二倍体状态。单倍体细胞通过减数分裂生成，基本过程是 DNA 复制一次而细胞分裂两次，因此每一个子代细胞仅含一套染色体。

有趣的是，虽然每一个细胞含全部基因，但在特定的细胞中仅有部分基因被用到(即被表达)。例如，肝细胞表达不同于皮肤细胞的一套基因。有机体的细胞如何分化成肝细胞、皮肤细胞等尚不清楚。 15

1.4.5 基因组类似计算机程序？

回顾蛋白质的产生过程很容易使人以“遗传程序”比喻该过程。在这个比喻中，生物的基因组被看做一个计算机程序，它完全规定了有机体，而细胞的功能仅是简单地解释该程序。由蛋白质完成生物功能就是执行这个“程序”。

这个比喻过于简化，原因包括以下两方面：

- DNA 程序实际上在转录和翻译中历经了改变，因此不能仅靠含基因的 DNA 片段的遗传密码来推断对应于该基因的蛋白质。
- 基因表达是一个依赖时空环境的复杂过程。例如，并不是所有的基因在生命周期中都表达，而某些基因可反复表达。部分基因仅在某些外部条件下表达，如病毒入侵；反之亦然，即在某些外部刺激下正常表达的基因可以被抑制。不过，一些基因的确基本上可自由表达，这正是生物技术成为可能的原因，但并非所有基因都能这样。如果我们把细胞内基因的表达看做“计算过程”，则可以说在人类基因组中，多达 10^{18} 个这样的过程在同时发生着并同时产生交互作用。

通过这些观察，看起来不应把生物体看做完全由其基因组决定，生物体实际上是一个复杂网络中各种因素同时相互作用的结果，基因组序列仅是起作用的因素之一。

1.5 基因组是如何被研究的

在基因组的研究中，首先要注意的是我们必须要面对不同尺度的问题。以人类基因组为例，从一段 DNA 片段中提取的基本信息是碱基对序列，获取该信息的过程叫做**测序**(sequencing)。人的一条染色体大约有 10^8 个碱基对。而另一方面，实验能够测序的最长 DNA 片段约为 700bp。这意味着在实际的测序和染色体大小之间大约有 10^5 的度量尺度的差异，这种差距也是计算生物学所面对的许多问题中的核心问题。在第 4 章(片段组装)和第 5 章(物理作图)我们将研究该类问题。在本部分中我们简要描述与这类问题有关的实验室技术。 16

1.5.1 图谱和序列

基因在染色体上的位置是非常重要的信息。术语"*基因座*"(gene locus)用来表明基

因在染色体上的位置。一个简单的问题是,给定的两个基因,它们是否为同源配对基因?无需借助于分子生物学技术,只要基因能够影响可视的性状即可判定,如眼睛的颜色与翅膀的形状。我们必须检测这些性状是否是独立遗传的。如果一个后代有 50% 的机会从一个亲代获得全部两种性状,我们就可以说它们是独立遗传的,或更技术性一点,这些性状独立地分配(assort)或分离(segregate)。如果性状独立地分配,则基因不连锁,即它们可能属于不同的染色体。而配对的基因应该一起分离,后代将可从亲代中继承两种性状。

事实上,生物学中的情况常常不是一清二楚的,并不总是有 100% 或 50% 的分离。由于交换(crossing over),各种百分比的分离都会发生。当细胞分裂产生子代细胞时,会形成新的基因排列,我们称做*基因重组*(gene recombination)。重组之所以发生是因为同源染色体在分离之前会交叉并交换它们的末端部分。存在无数种重组的可能性,所以我们实际看到的重组率变化很大,这些重组率给出了染色体上基因间距离的信息。如果基因紧密相连,因交换而分离的机会就很小;反之,如果距离远,分离的机会就增加,直至形成独立分配。

最初的遗传图谱是基于某些生物的,根据连续几代的数据而建立,并分析所观察到的某些性状的分离百分比。染色体的*遗传连锁图*(genetic linkage map)是一个显示基因间相对距离与次序的图。利用重组百分比构造的遗传图谱是重要的,但有两个缺陷:①它不能以碱基对或染色体上其他的线性长度单位告诉我们基因间的距离;②如果基因非常接近,我们就不能分辨它们的次序,因为分离的概率是如此小,所观察到的重组频率接近零。

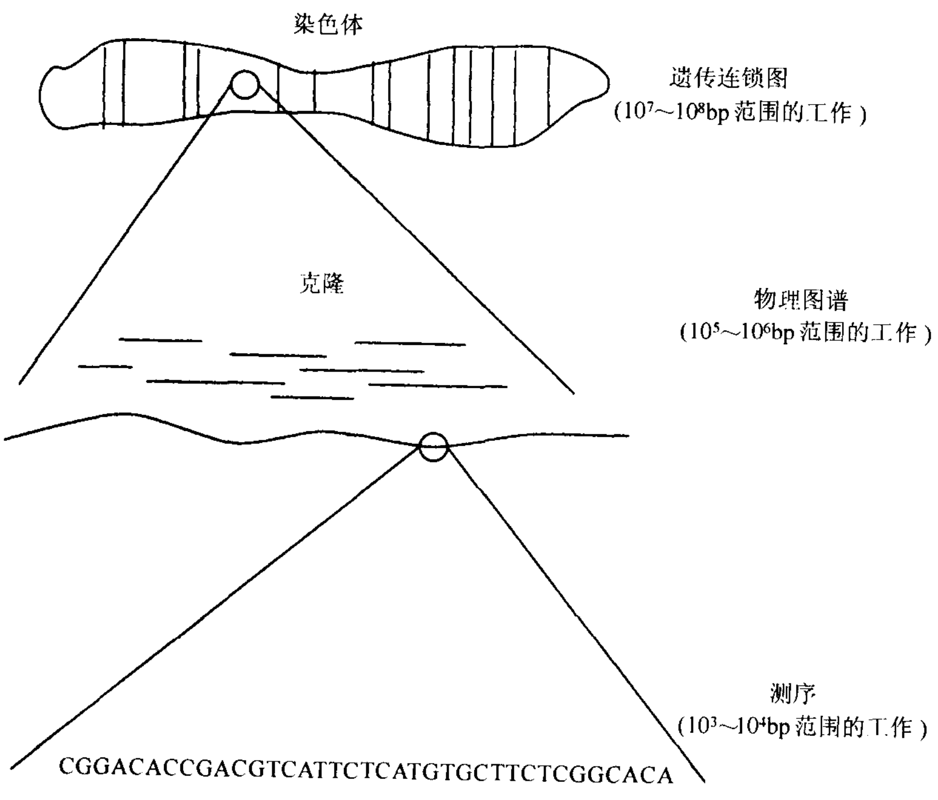


图 1.10 基因组研究的不同层次(基于文献[19]的一幅图)

以碱基对为单位反映实际距离的图谱称做**物理图谱**(physical map)。为建立物理图谱,我们需要完全不同的技术。特别是,我们不得不使用大大短于染色体但又长于可直接测序的 DNA 片段。物理图谱能够告知某些标记(marker)的位置,这些标记是已精确知道的小序列,长度小于 10^4bp。与物理图谱构造相关的计算问题将在第 5 章讨论。

最后,对于 10^3 数量级的 DNA 片段,我们可使用其他技术并获得全序列。我们曾提及当前的实验室技术至多只能测序 700bp 的片段。为测序20 000bp的 DNA 片段[所谓的**大规模测序**(large-scale sequencing)],一个基本策略是以不同的方式把序列打碎成几个拷贝,直接测小的序列,然后再使用在第 4 章中讨论的计算技术把它们组装起来。 17

图 1.10 显示了人类基因组研究的不同层次。

1.5.2　特殊技术

一些特殊的技术用于获得图谱和序列,在本部分中我们介绍其中的几个。

一个重要的问题是,须认识到分子生物学实验室技术所产生的数据几乎总是带有错误。由于这个原因,对于大多数计算生物学的算法,仅当它们能够处理错误时才是有用的。

= = = = = = = = = = = 病毒和细菌 = = = = = = = = = = =

我们从简短描述遗传学研究中广为使用的生物开始,它们是病毒和细菌。

病毒是分子水平的寄生虫。病毒难以被看做是一种生命形式,尽管它们在感染合适 18
的细胞后能够繁殖,被感染的细胞称做*宿主*(host)。病毒并不存在任何代谢,其本身无生化反应,因此,病毒依赖宿主细胞的代谢来复制自身,这也是其用于试验的原因。

大多数病毒由*蛋白帽*(衣壳,capsid)组成,其内包裹着遗传物质(DNA 或 RNA)。病毒 DNA 远远小于染色体 DNA,因此易于操作。当病毒感染细胞时,遗传物质进入细胞质,它们被宿主细胞错误地视做其自身的 DNA,因此细胞开始产生病毒编码的蛋白质,这些蛋白质促进病毒 DNA 的复制并形成新的衣壳,结果大量的病毒颗粒在宿主细胞中被组装。而且,一些病毒蛋白能够穿过细胞膜,在环境中释放新的病毒颗粒,攻击新的细胞。某些病毒并不立即杀死宿主细胞,而是其 DNA 可插入到宿主基因组中,长时间停留且不引起宿主细胞显著的变化。在特定的条件下这些静止的病毒能够被激活,从宿主基因组中释出并开始复制。

病毒具有高度的特异性,仅能感染特定的细胞。如 T2 病毒仅感染大肠杆菌;HIV,即人类免疫缺陷病毒,仅能感染一种防御外来微生物入侵的细胞;TMV,烟草花叶病毒,只感染烟草叶细胞,如此等等。噬菌体仅能感染细菌。

细菌是含一条染色体的单细胞生物,它可通过在短时间内简单地复制 DNA 而倍增,这使得它在遗传学研究中非常有用。实验室中最常见的细菌是大肠杆菌,它能够在 20min 内复制 1 次。由于其大小和复制速度,在实验室里很容易生成与处理极为大量的细菌。

= = = = = = = = = = = 切割和打断 DNA = = = = = = = = = =

因为 DNA 分子十分长,我们需要一些可在特殊位点将其切割(如同剪切)或按某种

方式将其打断的工具。这部分我们简要回顾用于这些过程的基本技术。

剪切的工具是**限制酶**(restriction enzyme),它们是能在特殊的位点催化 DNA 水解的蛋白质,这些特殊的位点叫做**限制位点**(restriction site),由局部碱基对决定。换言之,它们能够在所有出现特定序列的地方切割 DNA。例如,*Eco*RI 是一种能够在 GAATTC 序列出现的位置切割 DNA 的限制酶。需注意的是,这些序列本身反向互补,即$\overline{\text{GAATTC}}$=GAATTC。等于其反向互补序列的序列称为回文结构(palindrome)。因此,每当序列在一条链出现时,它同时也在另一条链出现。剪切发生在两条链的 G 和第一个 A 之间。结果,所产生的 DNA 具有“黏”端结构,也就是切点的 5′端比 3′端少 4 个碱基,如图 1.11 所示,这有助于和同一限制酶切割的另一个 DNA 片段再连接,提供一类“DNA 剪切与粘接”
19 技术,该技术在重组 DNA 的遗传工程中非常有用。

```
...ATCCAG|AATTCTCGGA...        ...ATCCAG        AATTCTCGGA...
...TAGGTCTTAA|GAGCCT...        ...TAGGTCTTAA        GAGCCT...
        DNA 剪切前                        DNA 剪切后
```

图 1.11　剪切 DNA 的酶,留下“黏”端

常见的限制酶产生长度为 4 个、6 个或 8 个碱基的切迹(cutter),奇数切迹的限制酶很少见,原因是奇数长度的序列难以形成回文结构。限制酶亦称做*核酸内切酶*(endonuclease),因它从内部切断 DNA,*核酸外切酶*(exonuclease)是指从末端向内降解 DNA 的酶。

在细菌中,限制酶起到抵御病毒攻击的作用,这些限制酶能够在病毒产生破坏之前切割病毒 DNA。另一方面,细菌 DNA 通过在某些碱基上增加甲基来使自身 DNA 免于被限制酶切割。

DNA 分子可通过*霰弹枪法*(shotgun method,又称鸟枪法)打碎成许多段,用于 DNA 测序。用含纯化 DNA 的溶液(其中含有大量相同的 DNA 分子)进行片段打断,例如通过剧烈的振动。每个 DNA 断裂成几个随机部分,某些部分被过滤后,被用于进一步的处理,特别是*复制*(copying)或*克隆*(cloning)。继而我们收集克隆的片段,测定这些片段,并组装成结果序列。这是确定原始纯化的 DNA 序列的标准方法。该方法亦可用于构造克隆文库,即覆盖某个长 DNA 分子的克隆集合。

==========复制 DNA==========

另一个在分子生物学研究中非常重要的工具是 DNA 的复制(或称 DNA 扩增,amplification)。我们面对的分子是非常小的对象,所以拷贝越多越容易研究。幸运的是,几种技术可实现 DNA 复制。

DNA 克隆(DNA cloning):在实验室做有关 DNA 的实验时需要最低量的 DNA,1 个分子显然不够。但有时开始时仅有一个分子。此外,这些材料必须储存起来,以供产生新的备份用于重复实验。DNA 克隆是达到此目的的一种技术。

20 给定一段 DNA 片段,获得更多拷贝的一种方式是利用自然界本身的能力。我们将

该 DNA 片段插入到一个宿主或载体(vector)的基因组中,然后让其自我复制。随着宿主的复制,插入的 DNA 片段(插入片段,insert)亦随着宿主 DNA 的复制而复制。然后,我们杀死宿主,丢弃其他 DNA,仅保留所插入的片段。这种产生 DNA 的方式叫做重组法(recombinant)。常见的载体包括质粒(plasmid)、黏粒(cosmid)、噬菌体(phage)和酵母人工染色体(yeast artificial chromosome,YAC)。

质粒是细菌内的环状 DNA 分子,它独立于细菌染色体且远小于细菌染色体,并随细胞的分裂而复制,每个子代细胞得到一个拷贝。质粒是理想的载体,但它对插入片段有大小上的限制,即约 15kb。插入大于此限制的片段需要大的质粒,但大的质粒常常在复制过程中变短。

噬菌体是用作载体的病毒。如噬菌体 λ,它可感染大肠杆菌。噬菌体中的插入片段可在其感染的宿主细胞后复制。噬菌体 λ DNA 的大小为 48kb,可以很好地耐受 25kb 的插入片段。插入更大的片段是不可能的,因形成的 DNA 太大,超过噬菌体包裹蛋白的适应范围。但如果整个噬菌体的 DNA 被插入片段所取代,并加上少许的复制装置,则 50kb 的插入片是可行的,这样的质粒称为黏粒。

对非常大的插入片段,如百万碱基对,可应用 YAC。YAC 是额外的、人工合成的染色体,通过在插入片段上加入酵母染色体控制区域而产生,看起来像是一个额外的染色体而能在酵母中得到复制。

聚合酶链反应(polymerase chain reaction,PCR):一种无需克隆即可获得大量 DNA 拷贝的方法是聚合酶链反应。DNA 聚合酶是一种能够催化单链 DNA 延伸的酶,延伸需要 DNA 模板,延伸的 DNA 与之配对,而互补的核苷酸添加到延伸链中直到两条链大小相等并形成正常双链 DNA 为止。聚合酶开始催化延伸时所需的那一小段双链 DNA 叫做**引物**(primer)。

PCR 包括相互交替的两个阶段:双链 DNA 受热变性而分成两条链,每一条单链通过添加引物和聚合反应变为双链。每一次重复使分子数量翻倍,经过足够的重复后产生足够数量的用于后续研究的 DNA。这种增长过程是指数性的。

一个有趣的题外话是,1983 年发现 PCR 的 Kary B. Mullis 认识到复制是个好主意,因为他"曾经花费大量时间书写计算机程序",非常熟悉迭代函数和指数生长。

＝＝＝＝＝＝＝＝＝＝＝阅读和测定 DNA＝＝＝＝＝＝＝＝＝＝

我们如何"阅读"DNA 序列中的碱基对?阅读是用一种称为凝胶电泳(gel electrophoresis)的技术完成的,该技术是基于分子可根据其大小而分离的原理。其过程涉及一个凝胶介质和一个强电场。DNA 和 RNA 分子在水溶液中带电,在电场力的作用下向 21
确定的方向移动。凝胶介质使它们移动速度减缓,速度与分子大小成反比。开始时所有分子置于凝胶块的一端,几小时后,小分子移向凝胶块的另一端,而大分子落在后面。通过插值法推算,分子的大小能够很精确地计算出来。

在这类实验中,DNA 分子可用放射性同位素进行标记,因此凝胶过程可被拍摄,产生电泳位置的图像记录。这一过程被用于 DNA 测序,确定限制片段的长度等还可用荧光染料替代同位素。可用激光束跟踪荧光染料并将信息直接输入计算机,连带完成图像处

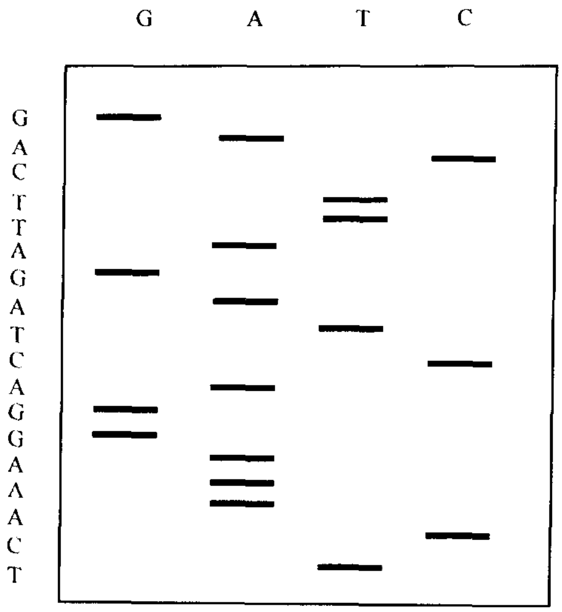


图 1.12　凝胶电泳产生胶片的图。单个 DNA 碱基能够通过 4 个列识别。短的片段近于顶部，长的片段近于底部

理。自动测序仪技术即基于此。

在这个过程中，可应用下面的技术阅读 DNA 或 RNA 的碱基。给定一个 DNA 分子，我们可获取终止于碱基 A 的各种片段，类似地，可获取终止于 T、G 和 C 的片段。使用 4 个不同的试管，对应于 4 种不同的碱基，每个试管中的片段大小不同。例如，假定原始的 DNA 片段是 GACTTAGATCAGGAAACT，则结束于 T 的片段分别是 GACT、GACTT、GACTTAGAT 和全序列本身。因此，如果我们能够按大小分离这些片段，且同时分别在 4 个试管中进行，就可以知道原始序列的精确碱基组成，如图 1.12 所示。

因标记可能模糊不清，特别是在接近胶片边缘处，在阅读凝胶片时易导致读片错误。另一个缺陷是用这种方式阅读的 DNA 片段长度有限，约为 700bp。

1.6　人类基因组计划

人类基因组计划是个多国合作项目，始于1988年，目标是完成人类染色体的全部物理图谱和全部 DNA 的测序。作为该计划的一部分，包括细菌、酵母、果蝇和小鼠等在内的其他生物的基因组也一并研究。

因人类基因组过于庞大，所以这不是一项容易的工作。迄今为止，许多病毒基因组已全部测定，它们的长度介于 1～10kb 之间。第一个完全测序的可独立生存的生物是流感嗜血杆菌(*Haemophilus influenzae*)，其基因组大小为1800kb。1996 年，酵母的全基因组序列的 10 000kb也已完成，这是一个非常重要的里程碑，因为酵母是独立生存的真核生物。人类基
22 因组图谱依然很遥远，因为其大小为目前已测定的最大基因组的 100 倍。

测定其他生物的全基因组序列作为人类基因组计划一部分的一个重要原因是完善测序技术以便将其用于人类基因组。随着新的改良技术的出现，测序费用有望降低。另一个原因与研究有关，因此所选定的生物都是广泛应用于遗传和分子生物学研究的。

人们已从人类基因组计划获得很多收益。如此规模的研究是任何一个实验室都无法单独完成的，因此一流实验室之间的协作可能是更有效的方式，甚至是惟一可行的方式。协作本身就是一项挑战，从计算机科学的角度来看，数据库需要更新，信息需要保持一致，数据需要能够被快速访问。一个主要问题是得到的序列中可能存在错误，目前的目标是每 10 000 个碱基允许出现 1 个错误。

另外一个问题是所谓的“平均基因组(average genome)”。不同的个体具有不同的基因组(这使 DNA 指纹成为可能)，一个基因在人群中可有许多等位基因，基因间隔序列

(并不编码任何蛋白质)在不同的个体之间也不相同。据估计,对两个不同的人,平均每500个碱基有1个碱基不同。因此存在这样一个问题:应测谁的基因组?即便选定了他或她作为测序对象,可转位因子(transposable element)也是一个困扰人的问题,因为已经知道基因组的某些部分可以从一个位置移动到另一个位置。所以,在测序之后,我们所能得到的至多也只是基因组的一个"快照"(snapshot)而已。

一旦获得全基因组序列,我们便面临艰巨的分析任务。我们必须识别出基因和确定 23
蛋白质的功能。基因识别仍处于萌芽阶段,而确定蛋白质的功能仍是一项非常繁重的工作。尽管令人鼓舞的先驱工作已获得结果,但离以人类基因组计划提供的数据来治疗遗传病距离尚远。

1.7 序列数据库

由于上节所述的技术和其他的因素,在过去几十年里人们已确定了大量DNA、RNA和蛋白质的序列,一些研究机构因此建立了储存这些序列和相关数据的数据库,新序列加入这些数据库的速度呈指数性增长,计算技术已被用于对这些数据库进行快速搜索,其中的一些将在第三章中叙述。这里我们简要叙述主要的有代表性的序列数据库。

GenBank:由美国国家生物技术信息中心(National Center for Biotechnology Information, NCBI)管理,含数万个DNA序列,根据不同的物种分为几个部分,包括:

- **PLN**:植物序列
- **PRI**:灵长类序列
- **ROD**:啮齿类序列
- **MAM**:其他哺乳动物序列
- **VRT**:其他脊椎动物序列
- **IVN**:非脊椎动物序列
- **BCT**:细菌序列
- **PHG**:噬菌体序列
- **VRL**:其他病毒序列
- **SYN**:合成序列
- **UNA**:未注释序列
- **PAT**:专利序列
- **NEW**:新序列

用户可用序列或关键词搜索GenBank。典型的入口项见图1.13。入口项由不同的域(field)组成,每个域有域标志及内容,域标志是一个描述该域内容的字。入口项是普通文本。一个重要的域是访问号accession no.(accession number),它是该入口项独有的代码,用来快速访问入口项。在该例子中,访问号是M12174。某些入口项有几个访问号, 24
它是几个相关但曾经独立的入口项综合成一个组合入口项的结果。

```
LOCUS       HUMRHOA      539 bp    mRNA          PRI      04-AUG-1986
DEFINITION  Human ras-related rho mRNA (clone 6), partial cds.
ACCESSION   M12174
KEYWORDS    c-myc proto-oncogene; ras oncogene; rho gene.
SOURCE      Human peripheral T-cell, cDNA to mRNA, clone 6.
  ORGANISM  Homo sapiens
            Eukaryota; Animalia; Chordata; Vertebrata; Mammalia;
            Theria; Eutheria; Primates; Haplorhini; Catarrhini;
            Hominidae.
REFERENCE   1  (bases 1 to 539)
  AUTHORS   Madaule,P.
  JOURNAL   Unpublished (1985) Columbia U, 701 W 168th St,
            New York, NY 10032
REFERENCE   2  (bases 1 to 539)
  AUTHORS   Madaule,P. and Axel,R.
  TITLE     A novel ras-related gene family
  JOURNAL   Cell 41, 31-40 (1985)
  MEDLINE   85201682
COMMENT     [2] has found and sequenced a family of highly
            evolutionarily conserved genes with homology to the
            ras family (H-ras, K-ras, N-ras) of oncogenes.
            [2] named this family rho (for ras homology).
            In humans at least three distinct rho genes are
            present. A draft entry and computer-readable copy of
            this sequence were kindly
            provided by P.Madaule (07-OCT-1985).

            NCBI gi: 337392
FEATURES             Location/Qualifiers
     source          1..539
                     /organism="Homo sapiens"
     CDS             <1..509
                     /note="rho protein;  NCBI gi: 337393"
                     /codon_start=2
BASE COUNT      105 a    180 c    172 g     82 t
ORIGIN      185 bp upstream of HinfI site.
        1 cgagttcccc gaggtgtacg tgc .... tatgtggccg acattgaggt
       61 ggacggcaag caggtggagc tgg .... ggccaggagg actacgaccg
      121 cctgcggccg ctctcctacc cgg .... atgtgcttct cggtggacag
      181 cccggactcg ctggagaaca tcc .... gaggtgaagc acttctgtcc
      ...
      421 cgaggtcttc gagacggcca cgc .... cgctacggct cccagaacgg
      481 ctgcatcaac tgctgcaagg tgc .... cgcgcctgcc cctgccggc
//
```

28

图 1.13　典型的 GenBank 人口项。除了最后一行，每个序列行固定有 60 个字符

这个数据库是国际协作的一部分，参与方包括日本 DNA 数据库 DDBJ 以及欧洲分子生物学实验室 EMBL。GenBank 的网址是：

http://www.ncbi.nlm.nih.gov/

EMBL:欧洲分子生物学实验室(European Molecular Biology Laboratary)是一个维护许多序列资源的研究机构,包括一个 DNA 数据库(核酸序列数据库 Nucleotide Sequence Database),其 25
组成类似于 GenBank,入口项主要域大体相同。在 EMBL 数据库中,每一入口项有一个双字母标记图 1.14,如访问号(accession number)标记为 AC。XX 表示空白行。GenBank 和 EMBL 都以 10 个字符为块表示序列,每一行有 6 个块,这种格式易于确定序列的特殊位置。EMBL 网址:

http://www.embl-heidelberg.de/

PIR:蛋白质识别资源库(Protein Identification Resource,PIR)是一个蛋白质序列数据库,由三方合作维护:美国国家生物医学研究基金、欧洲 MIPS(Martinsried Institute for Protein Sequence)和日本国际蛋白质信息中心。一个典型的入口项见图 1.15。主要网址:

http://www.gdb.org/

http://www.mips.biochem.mpg.de

PDB:蛋白数据库(Protein Data Bank)是蛋白质的三维结构数据库。对每一种蛋白质有一个一般信息头,然后是所有原子的列表,每个原子都带有其三维空间坐标表明它们在空间 26

```
ID   ECTRGA     standard; RNA; PRO; 75 BP.
XX
AC   M24860;
XX
DT   24-APR-1990 (Rel. 23, Created)
DT   31-MAR-1992 (Rel. 31, Last updated, Version 3)
XX
DE   E.coli Gly-tRNA.
XX
KW   transfer RNA-Gly.
XX
OS   Escherichia coli
OC   Prokaryota; Bacteria; Gracilicutes; Scotobacteria;
OC   Facultatively anaerobic rods; Enterobacteriaceae;
OC   Escherichia.
XX
RN   [1]
RP   1-75
RA   Carbon J., Chang S., Kirk L.L.;
RT   "Clustered tRNA genes in Escherichia coli: Transcription
RT   and processing";
RL   Brookhaven Symp. Biol. 26:26-36(1975).
XX
FH   Key             Location/Qualifiers
FH
FT   tRNA            1..75
FT                   /note="Gly-tRNA"
XX
SQ   Sequence 75 BP; 13 A; 24 C; 19 G; 19 T; 0 other;
     gcgggcatcg tataatggct attacctcag ... tgatgatgcg ggttcgattc
     ccgctgcccg ctcca
//
```

图 1.14　一个典型的 EMBL 入口项,除了最后一行,每个序列行固定有 60 个字符 27

```
PIR1:CCHP
cytochrome c - hippopotamus

Species: Hippopotamus amphibius (hippopotamus)

Date: 19-Feb-1984 #sequence_revision 19-Feb-1984 #text_change
      05-Aug-1994

Accession: A00008

Thompson, R.B.; Borden, D.; Tarr, G.E.; Margoliash, E.
    J. Biol. Chem. 253, 8957-8961, 1978
    Title: Heterogeneity of amino acid sequence in hippopotamus
      cytochrome c.
    Reference number: A00008; MUID:79067782
    Accession: A00008
    Molecule type: protein
    Residues: 1-104 <THO>
    Note: 3-Ile was also found

Superfamily: cytochrome c; cytochrome c homology

Keywords: acetylated amino end; electron transfer; heme;
    mitochondrion; oxidative phosphorylation; respiratory chain

Residues                Feature
1                       Modified site: acetylated amino end (Gly)
                          #status predicted
14,17                   Binding site: heme (Cys) (covalent) #status
                          predicted
18,80                   Binding site: heme iron (His, Met) (axial
                          ligands) #status predicted

                                Composition
           6 Ala  A       4 Gln  Q       6 Leu  L       2 Ser  S
           2 Arg  R       8 Glu  E      17 Lys  K       8 Thr  T
           5 Asn  N      14 Gly  G       2 Met  M       1 Trp  W
           3 Asp  D       3 His  H       4 Phe  F       4 Tyr  Y
           2 Cys  C       6 Ile  I       4 Pro  P       3 Val  V
     Mol. wt. unmod. chain = 11,530    Number of residues = 104

                5         10        15        20        25        30
  1 G D V E K G K K I F V Q K C A Q C H T V E K G G K H K T G P
 31 N L H G L F G R K T G Q S P G F S Y T D A N K N K G I T W G
 61 E E T L M E Y L E N P K K Y I P G T K M I F A G I K K K G E
 91 R A D L I A Y L K Q A T N E
```

28

图 1.15　一个典型的 PIR 入口项

结构中的位置。典型的例子见图 1.16 和图 1.17。这个数据库在布鲁克海文(Brookhaven)国家实验室,网址是:

http://www.pdb.bnl.gov

```
HEADER    MYOGLOBIN (CARBONMONOXY)                19-JUL-95   1MCY
TITLE     SPERM WHALE MYOGLOBIN (MUTANT WITH INITIATOR MET AND
TITLE    2 WITH HIS 64 REPLACED BY GLN, LEU 29 REPLACED BY PHE
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: MYOGLOBIN (CARBONMONOXY);
COMPND   3 CHAIN: NULL;
COMPND   4 ENGINEERED: YES;
COMPND   5 MUTATION: INS(MET 0), F29L, Q64H, N122D
SOURCE    MOL_ID: 1;
SOURCE   2 SYNTHETIC: SYNTHETIC GENE;
SOURCE   3 ORGANISM_SCIENTIFIC: PHYSETER CATODON;
SOURCE   4 ORGANISM_COMMON: SPERM WHALE;
SOURCE   5 EXPRESSION_SYSTEM: ESCHERICHIA COLI
KEYWDS    HEME, OXYGEN TRANSPORT, RESPIRATORY PROTEIN
EXPDTA    X-RAY DIFFRACTION
AUTHOR    T.LI,G.N.PHILLIPS JUNIOR
REVDAT   1   07-DEC-95 1MCY    0
JRNL        AUTH   X.ZHAO,K.VYAS,B.D.NGUYEN,K.RAJARATHNAM,
JRNL        AUTH 2 G.N.LAMAR,T.LI,G.N.PHILLIPS JUNIOR,R.EICH,
JRNL        AUTH 3 J.S.OLSON,J.LING,D.F.BOCIAN
JRNL        TITL   A DOUBLE MUTANT OF SPERM WHALE MYOGLOBIN
JRNL        TITL 2 MIMICS THE STRUCTURE AND FUNCTION OF
JRNL        TITL 3 ELEPHANT MYOGLOBIN
JRNL        REF    J.BIOL.CHEM.                  V. 270 20763 1995
JRNL        REFN   ASTM JBCHA3  US ISSN 0021-9258            0071
REMARK   1
REMARK   2
REMARK   2 RESOLUTION. 1.7  ANGSTROMS.
REMARK   3
REMARK   3 REFINEMENT.
REMARK   3   PROGRAM                         X-PLOR
REMARK   3   AUTHORS                         BRUNGER
REMARK   3   R VALUE                         0.182
REMARK   3   RMSD BOND DISTANCES             0.020    ANGSTROMS
REMARK   3   RMSD BOND ANGLES                1.82     DEGREES
REMARK   3
REMARK   3   NUMBER OF REFLECTIONS           23187
REMARK   3   RESOLUTION RANGE            5.0 - 1.7    ANGSTROMS
REMARK   3   DATA CUTOFF                     0.0      SIGMA(F)
REMARK   3
REMARK   3  DATA COLLECTION.
REMARK   3   NUMBER OF UNIQUE REFLECTIONS             25787
REMARK   3   RESOLUTION RANGE   INFINITY - 1.7        ANGSTROMS
REMARK   3   COMPLETENESS OF DATA                     95.8   %
```

图 1.16　一个典型的 PDB 入口项，含部分信息头，最后一列包含字符 1MCYi，i 为行号，此处略去。为适合页面要求，还做了一些其他编辑方面的改动　29

其他数据库：除上文所描述的之外，尚有其他一些保存分子生物学信息的数据库，最典型的是 ACEDB，一个 *C. elegans* 测序项目的数据库也适合简单的项目；Flyba se，一个果蝇

ATOM	1	N	MET	0	24.486	8.308	-9.406	1.00	37.00
ATOM	2	CA	MET	0	24.542	9.777	-9.621	1.00	36.40
ATOM	3	C	MET	0	25.882	10.156	-10.209	1.00	34.30
ATOM	4	O	MET	0	26.833	9.391	-10.078	1.00	34.80
ATOM	5	CB	MET	0	24.399	10.484	-8.303	1.00	39.00
ATOM	6	CG	MET	0	24.756	9.581	-7.138	1.00	41.80
ATOM	7	SD	MET	0	24.017	10.289	-5.719	1.00	44.80
ATOM	8	CE	MET	0	24.761	12.009	-5.824	1.00	41.00
ATOM	9	N	VAL	1	25.951	11.334	-10.816	1.00	31.10
ATOM	10	CA	VAL	1	27.185	11.834	-11.382	1.00	28.40
ATOM	11	C	VAL	1	27.330	13.341	-11.124	1.00	26.30
ATOM	12	O	VAL	1	26.444	14.135	-11.452	1.00	26.60
ATOM	13	CB	VAL	1	27.270	11.547	-12.912	1.00	29.30
ATOM	14	CG1	VAL	1	28.532	12.207	-13.526	1.00	28.60
ATOM	15	CG2	VAL	1	27.275	10.038	-13.163	1.00	29.70
ATOM	16	N	LEU	2	28.435	13.739	-10.500	1.00	23.00
ATOM	17	CA	LEU	2	28.691	15.142	-10.318	1.00	20.80
ATOM	18	C	LEU	2	29.289	15.737	-11.599	1.00	20.10
ATOM	19	O	LEU	2	30.129	15.110	-12.276	1.00	19.50
ATOM	20	CB	LEU	2	29.661	15.356	-9.134	1.00	20.90
ATOM	21	CG	LEU	2	29.036	15.387	-7.726	1.00	20.40
ATOM	22	CD1	LEU	2	28.556	13.983	-7.402	1.00	19.60
ATOM	23	CD2	LEU	2	30.058	15.904	-6.689	1.00	19.50
ATOM	24	N	SER	3	28.996	17.003	-11.826	1.00	18.80
ATOM	25	CA	SER	3	29.696	17.733	-12.852	1.00	19.40
ATOM	26	C	SER	3	31.096	18.141	-12.385	1.00	19.50
ATOM	27	O	SER	3	31.397	18.121	-11.174	1.00	19.10
ATOM	28	CB	SER	3	28.861	18.954	-13.223	1.00	20.60
ATOM	29	OG	SER	3	29.019	19.969	-12.261	1.00	22.10
ATOM	30	N	GLU	4	31.947	18.561	-13.310	1.00	18.40
ATOM	31	CA	GLU	4	33.293	18.956	-12.937	1.00	19.00
ATOM	32	C	GLU	4	33.173	20.215	-12.047	1.00	19.20
ATOM	33	O	GLU	4	34.026	20.457	-11.206	1.00	19.10
ATOM	34	CB	GLU	4	34.135	19.270	-14.198	1.00	19.60
ATOM	35	CG	GLU	4	35.491	19.937	-13.932	1.00	21.10
ATOM	36	CD	GLU	4	36.537	19.020	-13.295	1.00	22.60
ATOM	37	OE1	GLU	4	36.355	17.787	-13.230	1.00	23.90
ATOM	38	OE2	GLU	4	37.569	19.550	-12.840	1.00	24.60
ATOM	39	N	GLY	5	32.182	21.062	-12.313	1.00	18.30
ATOM	40	CA	GLY	5	32.039	22.287	-11.532	1.00	19.10
ATOM	41	C	GLY	5	31.669	22.025	-10.056	1.00	18.20
ATOM	42	O	GLY	5	32.251	22.638	-9.140	1.00	18.60

图 1.17　一个典型的 PDB 入口项,原子坐标值。最后一列包括字符 1MCYi,i 为行号,此处略
30 去。每行中移去了一些空格

的序列数据库;以及限制酶数据库和优先密码子数据库等。

习　题

1. 考虑序列 TAATCGAATGGGC,写出 6 个可能的蛋白质序列。
2. 对给定的下列基因片段,找到 a. 对应的 mRNA 序列,b.(所得到的)蛋白质序列(用标准遗传密码)
   ATGATACCGACGTACGGCATTTAA
   TACTATGGCTGCATGCCGTAAATT
3. 给定蛋白质序列 LMK,可能存在多少对应的 DNA 序列?(提示:其中一种可能是 CTGATGAAG)。
4. 找出限制酶 *Bam*HⅠ和 *Hin*dⅢ的切割模式。
5. 假设有一个长 40 000 的 DNA 分子,用一个 4-切迹限制酶消化。假设碱基在分子中是随机分布的,我们可能得到多少碎片?
6. 浏览 GenBank 数据库的病毒序列,找出一个你从未听说的致病病毒。

文 献 提 要

对分子生物学完全生疏的读者可能乐于读 Rosenfeld、Ziff 和 van Loon 的一本入门性的"漫画"书[165]。希望深入了解的读者可阅读分子生物学的标准参考书,其中一些是:

- Watson、Hopkins、Roberts、Steitz、Weiner 的著作。[203,204]
- Alberts、Bray、Lewis、Raff、Roberts、Watson 的著作。[7]
- Lewin 的著作。[124]

由于分子生物学的迅速发展,这些书一直在更新。

在准备这一章时,我们发现的其他有用教科书还有:生物化学方面有 Mathews 和 van Holde 的著作[131];遗传学方面有 Tamarin 的著作[183];蛋白质结构方面有 Branden 和 Tooze 的著作[28]。

第 1 页的引语出自 R.F.Doolittle 的著作[50],对于那些对蛋白质的了解是门外汉的读者,它是一篇好的综述。有关文献提供了关于 Mullis 发现 PCR 的有趣的介绍[14],第 15 31
页的引语亦出自该文。

Robbins[162]和 Frenkel[66]提供了从计算机科学角度讨论人类基因组项目困难性的文章。Robbins 强调了这样的观点,即一个生物是许多"细胞过程"的复杂交互的结果。Lewontin[125]对人类基因组项目提出了冷静的批评。

"遗传程序比喻"由许多人提出,我们的材料来自 Atlan 和 Koppel[17]。另一方面,对此感兴趣的读者可能乐于读一下 Hofstadter[96]的著作,它对于分子生物学和数理逻辑之 32
间的关联性进行了有趣的讨论。

第 2 章　串、图和算法

本章描述术语与约定，并简要介绍本书用到的数学与计算机科学知识。本章内容并不是一个完整的导论，对于未受这方面训练的读者，我们在文献提要中给出了推荐读物。

2.1　串

33 串是我们在本书中处理的基本数据结构。一个**串**(string)是**字符**(character)或**符号**(symbol)的有序排列，它们来自一个称做字母表(alphabet)的有限集合。序列(sequence)是串的一个同义词。书中大部分例子使用核酸字母表{A,T,G,C}或有 20 个符号的氨基酸字母表。但是，应注意到一些结论对所有字母表都适用。特别是，我们在这里所给的定义是普遍有效的。

串可以有重复字符，例如 s = AATGCA。串 s 的**长度**(length)定义为 $|s|$，是其所含符号的数量。在上例中，$|s|=6$。串 s 中第 i 个位置的字符为 $s[i]$，字符的下标从 1 至 $|s|$。在本例中，我们有 $s[1]$ = A，$s[2]$ = A，$s[3]$ = T，等等。长度为 0 的串称为空串(empty string)，用特殊符号 ε 表示。

尽管串和序列意义相同，**子串**(substring)和**子序列**(subsequence)却是不同的概念。s 的子序列是一个序列，可通过移去 s 中(任意)若干字符得到。例如，TTT 是 ATATAT 的子序列，但 TAAA 不是。空串是任何序列的子序列。当序列 t 是序列 s 的一个子序列时，我们称 s 是 t 的一个**超序列**(supersequence)。

与子序列概念不同，s 的子串是一个由 s 中的连续字符组成的串，且保持它们在 s 中
34 的顺序。例如，TAC 是 AGTACA 的子串，但 TTGAC 却不是。串 s 的子串同时也是序列 s 的子序列，但序列 s 的子序列未必是串 s 的子串。当序列 t 是序列 s 的子串时，我们称 s 是 t 的一个**超串**(superstring)。

有时，u 的子串 w 在 u 中出现多次，如当 w = TT 而 u = CTTTAGCATTAA 时。在某些情况下区别这些出现是重要的，为此我们使用**区间**(interval)的概念。串 s 的一个区间是一个连续下标集 $[i\cdots j]$，且 $1\leqslant i\leqslant j+1\leqslant |s|+1$。区间包含 i 与 j 间的所有下标，包括 i 与 j 本身。对于 $[i\cdots j]$，s 的一个区间 $s[i\cdots j]$ 表征 s 的一个子串 $s[i]s[i+1]\cdots$

$s[j]$,若 $i \leqslant j$。当 $i=j+1$ 时这是个空串。因此对 s 的任何子串 t,存在至少一个 s 的区间 $[i \cdots j]$ 使得 $t=s[i \cdots j]$。

串 s 和 t 的**拼接**(concatenation)由 st 表示,将 t 中的全部字符按它们在 t 中的次序续于 s 之后。例如,如果 $s=$ GGCTA 且 $t=$ CAAC,则 $st=$ GGCTACAAC。s 和 t 的长度是 $|s|+|t|$。串 s 几个相同拷贝的拼接用 s 的幂表示,如 $s^3=sss$。

s 的一个**前缀**(prefix)是 s 的形如 $s[1 \cdots j]$ 且 $0 \leqslant j \leqslant |s|$ 的任意子串。$j=0$ 是合法的,且定义 $s[1 \cdots 0]$ 为空串,它也是 s 的一个前缀。注意 t 是 s 的一个前缀当且仅当存在另一个串 u 使得 $s=tu$。有时我们需要引用含恰好 k 个字符的 s 的前缀,$0 \leqslant k \leqslant |s|$,这时用记号 $prefix(s,k)$ 指称这种串。

类似地,s 的一个**后缀**(suffix)是一个形如 $s[i \cdots |s|]$ 的子串,$1 \leqslant i \leqslant |s|+1$。$i=|s|+1$ 亦合法,这时 $s[|s|+1 \cdots |s|]$ 是空串。串 t 是 s 的一个后缀当且仅当存在 u 使得 $s=ut$。记号 suffix(s,k) 指称 s 的含 k 个字符的后缀,$0 \leqslant k \leqslant |s|$。

杀手因子(killer agent):所有的串 s 满足 $|s| \geqslant 0$。为简化自变量,我们假定存在一个特殊的串 κ 使得

$$|\kappa| = -1$$

这个串起"杀手因子"作用,即删除与其紧邻的字符。例如

$$\kappa \text{GCTAGT} = \text{CTAGT}$$

注意所得结果是原始串的一个后缀。一般地,带 κ 的幂的拼接可被用于指称前缀、后缀、和子串。

$$Prefix(s,k) = s\kappa^{|s|-k}$$

$$Suffix(s,k) = \kappa^{|s|-k}s$$

$$S[i \cdots j] = \kappa^{i-1}s\kappa^{|s|-j}$$

使用杀手因子有一个缺陷,即拼接不再是可结合的。例如

$$(\text{ATC}\kappa)\text{GTC} = \text{ATGTC}$$

而

$$\text{ATC}(\kappa\text{GTC}) = \text{ATCTC}$$

因此若不带括号,ATCκGTC 的意义是不确定的。但是,我们仍然可有等式 35

$$|st| = |s| + |t|$$

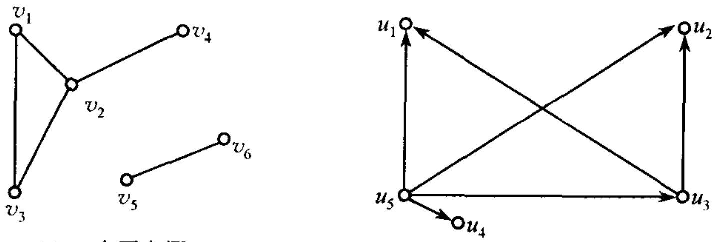
这里 s 和 t 是可带 κ 的一般串表达式。

2.2　图

一个**图**(graph)可用两个集合表示:V 是结点(或顶点)集,E 是边集。一个图可由 $G=(V,E)$ 表示,顶点由字符 u,v,w 表示,边由字符 e 或顶点对 (u,v) 表示。图可以是无向的,此时边是顶点的无序对[因此 $(u,v)=(v,u)$];或有向的,此时边是顶点的有序对[因此 $(u,v) \neq (v,u)$]。若无特别说明,本书后续章节中的图均指无向图,而本章中不带说明的图则兼指有向图与无向图。尽管图是抽象的数学对象,但它们通常可用线条画出,如图 2.1 所示。除非额外说明,本书中的所有图都是简单图,即它们不包含形如

(u, u)的回路，同一对顶点间也不包含多条边[注意，在有向图中(u, v)与(v, u)是不同的边]。顶点的数量用$|V|$或 n 表示，而边的数量用$|E|$或 m 表示。

给定边(u, v)，我们说 u 和 v 是它的端点，u，v 与(u, v) 关联，反之亦然。如果(u, v)是有向边，则 u 是边的尾，v 是边的头。当 u 和 v 是一个无向边的端点时，我们说 u 和 v 是邻接的。在一个无向图中，顶点 v 的度指的是与其邻接的顶点的数量。在有向图中，一个顶点 v 有一个出度和一个入度，出度指的是形如(v, x)的边的个数，入度指的是形如(x, v)的边的个数。例如图 2.1 中，顶点 v_2 的度是 3，v_5 的度是 1，u_3 的入度是 1，出度是 2。



(a) 一个无向图　　(b) 一个有向图

图 2.1　图的例子

36

在一个加权图中，每个边与一个实数相关联。根据上下文，边(u, v)的权可称做代价，或 u 与 v 之间的距离，或就称为权。

给定图 $G = (V, E)$及 $G' = (V', E')$，我们说 G'是 G 的一个子图，如果 $V' \subseteq V$ 且 $E' \subseteq E$。如果 G'是 G 的一个子图但$G' \neq G$，我们就说 G'是 G 的真子图。如果 G'是 G 的一个子图而 $V' = V$，我们说 G'是 G 的一个生成子图。对于给定 E'，如果 V'是关联于 E'中某边的全部顶点的集合，我们说 G'是 E'导出的图。反之，对给定 V'，如果 E'是两个端点均在 V'中的边的集合，我们说 G'是 V'导出的图。

图中的一个通路是一个顶点系列$(v_1, v_2, \ldots, v_k)$，满足(v_i, v_{i+1})是图中的边且 $1 \leq i \leq k$。有向图中的一个环是一个通路，满足 $k > 1$ 和 $v_1 = v_k$；无向图中的一个环也是一个通路，满足 $v_1 = v_k$，且无重复边。除非另有指明，否则所有环均为简单环，它意味着除了开头和最后的顶点所有的顶点都各不相同。一个简单环的例子是图 2.1 无向图中的(v_1, v_2, v_3, v_1)。我们说顶点 v 是由顶点 u 可达的，如果存在一条 u 与 v 之间的通路。在加权图中，一个通路的权是其各边的权之和。

在一个无向图 G 中，如果每个顶点都是由任意其他顶点可达的，则这个图是连通的。如果图不是连通的，我们可以发现它的一个连通成分，连通成分的概念是直观的。例如，图 2.1(a)是一个带两个连通成分的无向图。更形式化的定义是，一个图 G 的连通成分集是 G 的所有连通子图的集合，其中没有一个元素是另一个元素的子图。对一个有向图，它可能是强连通、弱连通或不连通的。对一个强连通图，每个顶点都是从任意其他顶点可达的；对一个弱连通图，仅当我们忽略边的方向时才成立，图 2.1(b)的有向图就是这样一个例子。如果这两个性质均不满足，图就是不连通的。

图理论根据某些特性对图进行分类，一些重要例子有：

- 无环图：不带环的图。

- 完全图:对每一对顶点 v 和 w 都有 $(v,w)\in E$[若是有向图则同时也有 $(w,v)\in E$]。
- 二部图(偶图,bipartite graph):顶点分为不相交的两个子集 U 和 V,且任何边的两个顶点都分别属于 U 与 V。
- 树(tree):树是个无环连通图。一个连通成分均为树的图称森林(forest)。

树是图的重要一类,有它们自己的术语。叶子(leaf)是树的度为 1 的结点,所有其他结点称内部结点。树可以有根,即有一个称为根的特殊结点(表示为 r)。在从结点 u 到根 r 的通路上,对结点 u 和 v,称 v 是 u 的一个祖先,u 是 v 的一个后继;如果 u 和 v 是邻接的,则 v 是 u 的双亲(parent),u 是 v 的孩子(child)。叶子是无孩子的结点,而根则无祖先。结点 v 的深度是从 v 到 r 的通路上边的个数。两个结点 u 和 v 的最低公共祖先(the lowest common ancestor)是 u 和 v 的最深的祖先结点。 37

计算生物学中另一类重要的图是区间图(interval graph)。一个区间图 $G=(V,E)$是一个无向图,它由实线上区间的集合 C 而来。C 中的每一个区间对应于 G 中的一个顶点。我们在顶点 u 与 v 间置一条边,当且仅当 u 和 v 对应的区间之间有一个非空交集。

下面我们列出几个与本书密切相关的有关图的重要问题:

1. 给定一个有向或无向图,找出一个环使得图中每一条边都在环中出现且仅出现一次(顶点可以重复),这样的图称欧拉图(Eulerian graph)。这个问题的另一种表述是在图中发现一个满足同样条件的通路[该图因而也称包含一条欧拉通路(Eulerian path)]。
2. 给定一个有向或无向图,找出一个环使得图中每个顶点都在环中出现且仅出现一次(除了最初和最后的顶点),这样的图称哈密顿图(Hamiltonian graph)。这个问题的另一种表述是在图中发现一个通路,使得图中每个顶点都在通路中出现且仅出现一次[该图因而称包含一条哈密顿通路(Hamiltonian path)]。还有一种表述是定义在有权图上的,即找出一个权值最小的哈密顿环,这也称为货郎担问题(旅行售货员问题),城市如同顶点,道路如同边,售货员需要以最短的距离遍历所有城市。
3. 给定一个无向带权连通图,找出一个生成树,使得其所有边的权的和在所有生成树中是最小的,该问题称做最小生成树问题。
4. 给定一个无向图,找出着色该图的最小颜色数量。图的顶点着色是一种功能,它把整数(颜色)赋值到顶点且没有相邻顶点具有相同颜色。
5. 给定一个无向图,找出一个最大基数匹配。一个匹配是边的一个子集 M,其中没有两个边共享同一端点。这个问题在带权图中的表述是找出一个总权值最大的匹配。

在计算机程序中图能够用许多方法表示,最常用的方法是邻接矩阵表示法(adjacency matrix representation)。给定一个 $n\times n$ 的矩阵,如果$(i,j)\in E$ 则令 $M_{ij}=1$,否则 $M_{ij}=0$。如果图是带权的,我们用边的权代替矩阵中的 1。这种表示需用 n^2 的存储空间。另一种表示图的方法是邻接表表示法。在这种方法里,我们在一个表中存储所有的顶点,然后对表中的每一个顶点创建另一个表,表要包含该顶点的所有邻接顶点。在有向图的情况下,每个顶点有两个表,一个用于所有从 v 发出的边,另一个用于所有进入 v 的边。这种表

38 示使用 $n+m$ 的存储空间。当图中边的数量较少时，邻接表相比邻接矩阵具有节省空间的优点，这样的图称为稀疏图。当 m 显著大于 n 时，我们就说图是稠密的。

2.3 算法

一般地讲，算法是一个明确定义的有限步骤序列，用于求解一个明确定义的问题。有个例子就是两个整数相乘的步骤序列。算法是由人或机器执行的。对于规定何人或何物执行算法是相当重要的，因为这与“明确定义”的寓意所在有关。本书中的算法是可被人理解和被机器执行的。我们所指的“机器”不是具体的机器而是一种抽象，称做随机访问机器模型。简单地说，这个机器包括下列成分：一个能顺序执行算法步骤的处理器，有限的内存（字的集合），有限的寄存器集合（字的另一个集合）。每个字存储一个整数或实数，被处理器执行的程序是用机器的指令编写的算法。在本书中我们对机器指令一词的使用并不严谨，而易被人理解的非形式的指令若要在真正的计算机上执行尚需做很大的扩展。

本书中的算法同时用英语语句和伪程序设计语言代码书写，使用这种伪代码的目的之一是方便想实现算法的读者。我们并不形式地定义这种代码，而只是用 C 或 Pascal 这样的程序设计语言简单解释之。首先，我们描述记号约定：

1. 算法由关键字 Algorithm 开头。
2. 关键字 input 和 output 用于说明算法的输入和输出。
3. 关键字用黑体打印。
4. 变量、过程、函数名用斜体打印。
5. 常数用小大写字母打印。
6. 注释以“//”开头，直至一行结束。
7. 非形式语句和条件以正常字体打印。
8. 不同的语句通常写在不同行，分号用于分离同一行的语句。当一个非形式的语句需要书写一行以上时，第二行开头缩排。
9. 程序块以缩排开始，不带 begin 和 end。偶尔使用注释标明长程序块的结束。
10. 阵列的下标用方括号表示（如 $V[i]$），记录的下标用点号表示（如 $R.field$）。

39 11. 过程或函数调用括号中的参数以逗号分隔。

下面我们描述算法中使用的形式语句，它们大多数与现代过程式语言中的相应语句类似。

1. 赋值语句，形如 $a \leftarrow a+1$。
2. for 迭代语句，形如 for $i \leftarrow 1$ to n do。其变体包括关键字 downto 以及非形式项中迭代的表示（形如 for 每个元素 of 集合 S do）。
3. 条件语句 if 〈条件〉 then...else ...。
4. 迭代语句 while 〈条件〉do...。
5. 迭代语句 repeat...until 〈条件〉。
6. 中断语句 break，用于中断最内层迭代语句的执行。
7. 返回语句 return，用于中断函数的执行，返回一个值或结构给其调用者。

语句中的〈条件〉可以是形式的或非形式的。

"算法"意味着能求解所针对问题的所有实例,正确的算法应能满足这一条,因此算法的正确性证明就十分重要。算法的另一个重要方面是运行时间,它指的是算法在求解一个特定实例时所需执行的机器指令数。给出正确性证明和估计运行时间是算法分析的重要方面。

之所以估计算法的运行时间而不是计算指令的确切数量,是因为我们想知道算法在求解任意问题实例时的算法行为。为获得这样的信息,我们需要使用两条规则。第一,算法运行时间指的是最坏情况下的运行时间;第二,运行时间依赖于所求解实例的规模。因此,运行时间不是一个数而是一个函数。例如,我们可以说算法 A 的运行时间是 $5n^2+3n+72$,n 是问题规模的量度。运行时间通常被简化,去掉常数和低阶项。在上例中,相对于 n^2,$3n$ 和 72 是低阶项,因此 A 的运行时间是 $O(n^2)$,记号 O 表示我们不计较常数和低阶项。因而,我们可以根据算法的渐进行为对算法进行比较。例如,对于一个小的 n,一个 $O(n^2)$的算法可能比一个 $O(n)$的算法还要快,但当 n 足够大时 $O(n)$肯定更快。相应地,具有 $O(n)$运行时间的算法称为线性的,而具有 $O(n^2)$的算法称为平方的。

从上面的讨论可看出,为了比较算法的运行时间,我们需要比较函数的增长速率,即给定两个从整数域到实数域的函数,哪个增长更快?我们假定读者能够利用下面的形式定义比较简单函数。给定两个从整数域到实数域的函数 $f(n)$和 $g(n)$,如果存在常数 c 和 n_0 使得对所有 $n>n_0$,$g(n)\leqslant cf(n)$,则 $g(n)=O[f(n)]$。换言之,当我们知道函数 $g(n)$增长不快于函数 $f(n)$时,就有 $g(n)=O[f(n)]$,f 是 g 的一个上界。例如,当 40
$g(n)=\log n$,$f(n)=\sqrt{n}$时上式就成立。假如我们不计较常数项,对数的底(它此时是一个常数)在 O 内就是无关的,因为当 a 和 b 是常数时,$\log_a n$ 与 $\log_b n$ 只差一个数乘因子。

有时,我们需要把函数 $g(n)$描述成至少和某个函数 $f(n)$增长一样快,这时我们说 $g(n)=\Omega[f(n)]$ 。形式上,这意味着存在常数 c 和 n_0 使得对所有 $n>n_0$,$g(n)\geqslant cf(n)$。函数 f 为 g 提供了一个下界(lower bound)。当 $g(n)$既是 $O[f(n)]$也是 $\Omega[f(n)]$时,我们说 $g(n)=\Theta[f(n)]$。

在本书中我们只对有效的算法感兴趣。对某个规模为 n 的问题,当一个算法具有多项式的运行时间时我们称它是有效算法。一个运行时间为 $\Omega(2^n)$的算法不是有效的,因为函数 2^n增长快于任何 n 的多项式。

理论计算机科学家的主要任务之一是把各种问题分为两类:存在有效算法的问题(P 类问题)和不存在有效算法的问题,这一任务称做复杂性分析。不幸的是,存在一类非常重要的至今尚未能判定其分类的问题,对它们既未发现有效算法,也未证明有效算法不存在。这类问题称做 **NP-完全问题**(NP-complete problem),它们属于 NP 类,而 NP 类问题的解法一经被发现可在多项式时间内被检验。P 类问题是 NP 类问题的一个子集。更准确地讲,NP-完全问题是 NP 类问题中最艰巨的问题,即 NP 类中任何问题的任何实例可在多项式时间里转换为 NP-完全问题。也可以说,所有 NP-完全问题在多项式变换下是等效的。这意味着如果一个 NP-完全问题发现了一个多项式算法,那么所有的 NP-完全问题均能在多项式时间内求解。

上面提到的未判定问题就是 P=NP 问题,字面上,它指是否所有 NP 问题均可在多

项式时间内求解？如果 P＝NP，则对每一个 NP 问题均存在有效算法，特别是包括 NP-完全问题。如果 NP≠P，则所有 NP-完全问题都是难以计算的，P 是 NP 的真子集。大多数计算机科学家相信后一结论。

NP-完全问题频繁出现于本书，而另一个相关的术语是 **NP-难题**（NP-hard problem）。要解释它们之间的区别我们就必须对问题类做更精确的陈述。严格地说，P 和 NP 类问题仅由判定问题组成：答案或为是或为否。一个典型例子是判定一个图是否是哈密顿图。然而，本书中遇到的大部分问题是优化问题，即存在某个与问题相关的函数，我们要找出一个使函数值最小化或最大化的解法。一个典型例子是 2.2 节中的旅行售货员问题。引入一个输入值 K，并设问是否存在一个值小于 K（最小化问题）或大于 K（最大化问题）的解，一个优化问题就可以转换为一个判定问题。这个技巧使得我们能够提出与每个优化
41 问题对应的判定问题，然后试图对它们进行分类，属于 P 或 NP。由此可见旅行售货员问题的判定问题版本是一个 NP-完全问题。但是该问题本身如何？因为它不是一个判定问题，故它不属于 NP 类，但显然它至少和它的判定问题版本一样困难。至少和 NP-完全问题一样困难但不属于 NP 类的问题称为 NP-难题。所有的优化问题，如果其判定问题版本是 NP-完全问题，均是 NP-难题。

对 2.2 节中的问题，下面这些结论是已知的。要判定一个图是否是欧拉图是一个 P 类问题，如果它能在 $O(n+m)$ 时间内求解；要判定一个图是否是哈密顿图是一个 NP-完全问题，因为已知旅行售货员问题是 NP-难题。一个最小生成树可在时间 $O(n^2)$ 内被发现，因此（它的判定问题版本）是 P 类问题。顶点着色问题是另一个 NP-难题。最后，匹配问题可在多项式时间内求解。

当我们初次遇到一个复杂性不明的问题 X 时，可有两个选择：试图找出一个有效求解 X 的算法（即试图证明该问题属于类型 P），或试图证明该问题是 NP-完全的。对后一任务，一个非常简要的步骤描述是，首先应证明问题属于 NP 类；（它的解，如果存在的化，可否在多项式时间内被检验？）然后应证明一个已知的 NP-完全问题 Y 可约简为 X，这可通过证明下面这个命题得到：如果我们对问题 X 有一个多项式时间的算法，则我们亦可用其在多项式时间内求解 Y。发现一个能被简单地约简为 X 的 NP-完全问题 Y 通常是 NP-完全性证明的最难部分，不过，已有超过一千个问题已被证明是 NP-完全的，这一事实已使该任务变得容易多了。

当遇到一个 NP-完全问题或 NP-难题时该怎么办？这里是几个选项：

1. 检查待求解问题的实例是否一般化。在某些情况下，针对受限的输入，求解一个 NP-完全问题可在多项式时间内完成。一个经典例子是图的着色。对一个一般的图发现一个最小着色是 NP-难题，但如果输入是一个双向图，则该问题可在多项式时间内求解（任何双向图可用至多两种颜色着色）。
2. 设计一个穷尽搜索算法（即枚举所有可能解得出其中最佳者）。这通常仅对小的实例适用，因为这样算法的运行时间是指数的。
3. 设计一个多项式时间的近似算法，它能得到一个保证接近优化的解，这种保证可表达为近似解与优化解之间的比率。例如，对旅行售货员问题的某个版本存在一个近似算法，其得到的近似解的权值不大于优化解的 1.5 倍。这种保证是否有实

际意义取决于保证本身以及实际问题。

4. 设计启发式算法,这些是多项式时间算法,但对解的质量不做保证。在许多情况下启发式算法能满足实际性能需要,因而广被使用。 42

我们现在讨论另外几个算法已知的重要问题。第一个是检验一个无向图是否是连通的,这个问题可用著名的图遍历技术解决。技术之一是深度优先搜索,由任意顶点 v 开始,访问 v,然后访问未访问的邻接点。重复这一过程直至或者所有顶点都被访问,或者终止(所有可达的顶点均已访问,但仍存在未访问的顶点)。第一种情形图是连通的,第二种情形则不是。另一个技术是广度优先搜索,使用一个队列存储顶点。我们首先访问队列中第一个顶点的所有邻接顶点,并把它们放入队列中以待后续处理,然后从队列中取出下一个顶点访问之,依次进行。两个算法的运行实际均为 $O(n+m)$。这些技术可被用于许多其他图问题中,尤其是深度优先搜索可被用于检验一个图是否为无环图。树亦可进行遍历,虽然理所当然是连通的,但用这里的技术可系统地访问其所有结点。

最小生成树(minimum spanning tree,MST)问题可被有效求解,所用技术要比简单遍历精致些。对一个有 n 个结点的图,我们能用一个时间为 $O(n^2)$的简单算法发现一个 MST,其过程如下。从任意顶点开始,通过加入连接树中结点与非树中结点的最小权的边来扩展树。这个简单算法能够奏效的原因是我们总是依据“局部”最佳原则选择边。使用这种技术的算法称做贪婪算法。

P 类问题中另一个问题是排序。给定一个有 n 个不同数字的向量,按升序或降序排列它。可以证明如果数字是任意的,排序需 $\Theta(n\log n)$的时间。如果数字界于某常数内,或界于某固定度的多项式(如 n^3),则问题可在线性时间内求解。

结束本节之前,我们介绍几个算法设计的概念和工具。第一个有用的概念是归纳。给定一个问题,其实例的规模为 n,我们首先找出该问题的小规模实例的解法(例如如何对一个数进行排序),并假定我们已知如何求解规模 $n-1$ 的实例,然后考察如何把规模 $n-1$ 的实例的解扩展到规模 n 的解。在许多情况下这种扩展是简单的。数学归纳可证明我们已一般性地解决了某个问题。当使用这种技术设计算法时,结果常常用递归表示。在程序设计语言里,一个递归函数是一个调用其自身的函数(直接或间接)。图 2.2 刻画了这个概念。 43

```
Algorithm RSort
    input: array A with n distinct elements
    output: array A sorted in increasing order
    if n = 1 then
        return A
    else
        x ← min(A)   // min returns the smallest element
        Remove x from A
        RSort(A)   // recursive call
        return x : A   // we concatenate x to the beginning of A
```

图 2.2　对 n 个元素排序的递归算法

本书中一个特别重要的程序设计技术称为*动态程序设计*(dynamic programming)，其基本概念是利用许多小而相似的子问题的解来建立问题本身的解，它可被看做归纳法的一个特例，具体的例子在第三章中。注意这个技术与计算机程序设计之间没有任何关系，它是由于历史的原因而得名。

数据结构是算法设计中用到的另一个工具。实际存在各种存储数据的结构，高效地支持某些操作，如检索/插入等，操作时间为常数或 $O(\log n)$，n 是所存储数据的项数。数据结构的类型依问题本身而定，最常用的数据结构是一些特殊类型的表，包括栈、队列、hash 表、二叉搜索树等。不熟悉这些结构的读者可参考文献提要中的相关论著。本书中用到的一个不太熟知的数据结构是*非相交集森林*(disjoint-set forest)，它支持对不相交集进行动态汇集的操作。初始时，每个数据项自身都是一个集合，随着算法的执行，集合逐渐彼此融合。这个数据结构支持三个基本操作：*MakeSet*(a)，它把 a 放入仅由它自己构成的集合；*FindSet*(a)，返回包含 a 的集合，*Union*(S,T)，融合集合 S 与 T。缘于此，支持这种操作的数据结构称为融合-查找数据结构(union-find data structrue)。如果我们总共有 n 个单元，一个非相交集森林结合其他两种技术可使一系列 $m>n$ *FindSet* 和/或 *Union* 操作在时间 $O[m\alpha(m,n)]$内完成，这里 $\alpha(m,n)$是一个增长极其缓慢的函数，以致对所有实际问题它可作为常数看待。

习　　题

1．ACTAC 有多少个不同的子串？多少个不同的子序列？多少个区间？

2．设计并分析一个算法，检查串 p 是否是另一个串 t 的子串。能否设计一个运行时间为 $O(|p|+|t|)$的算法？

3．用你最熟悉的计算机语言写一个程序，接收任意长度的 DNA 序列作为输入，返回它的逆序。

4．一个 n 个顶点的完全有向图有多少条边？如果图是无向的，它有多少条边？一个 n 个结点的树有多少条边？

5．证明对无向图 $G=(V,E)$，存在 $\sum_{v\in V}\delta(v)=2|E|$，$\delta(v)$是顶点 v 的度。

44 6．给出图 2.1 中图的邻接矩阵和邻接表表示。

7．解释对小的 n 值，一个 $O(n^2)$算法为什么可能比一个 $O(n)$的算法更快。

8．假如对同一问题你有两个算法 A_1 和 A_2，问题的规模由参数 n 定义。如果 A_1 的运行时间是 $O(n/\log n)$，A_2 的运行时间是 $O(\sqrt{n})$，哪个算法更快？

9．欧拉证明一个图是欧拉图当且仅当它是连通的且所有的顶点有相同的度。基于这一点，提出一个在图中找出欧拉环的算法，假定该环存在。

10．证明对一个所有的边代价(cost)各不相同的有权无向图，其最小生成树是惟一的。

11．基于书中给出的信息，设计一个最小生成树问题的完整算法，并分析你的算法。

12．一个有向图的拓扑排序(topological ordering)指的是对所有的边(u,v)，u 出现于 v 之前。拓扑排序仅对有向无环图有良好定义。设计和分析一个算法，确定一个有向无环图 G 的拓扑排序。

13. 假如你有一个非相交集森林数据结构，使用它设计一个算法确定一个无向图 G 的连通成分。将你的算法与另一个基于深度优先或广度优先的算法相比较。
14. 假定你有一个复杂性不明的问题 X，你成功地将它归约为一个已知的 NP-完全问题。对应 X 的复杂性你有何种结论？
15. 顶点覆盖问题(vertex cover problem)定义在一个无向图 $G=(V,E)$上，寻找一个顶点的最小集 $W\subseteq V$ 使得对任何边 $e\in E$ 至少一个端点属于 W。写出这个问题的判定问题版本。
16. 图 G 中的一个系(团)(clique)是 G 的一个完全子图。系问题定义在一个无向图 $G=(V,E)$上，寻找一个顶点的最大集 $C\subseteq V$ 使得 C 是一个系。由于顶点覆盖问题是 NP-完全问题，证明系问题也是 NP-完全问题。
17. 假定哈密顿环问题是 NP-完全问题，证明两给定顶点间的哈密顿通路问题是 NP-完全的。
18. 顶点覆盖问题的一个近似算法如下。从一个空覆盖 W 开始，选择任意边(u,v)并增加 u、v 到 W，然后从 G 中删除 u 和 v(即移去任何与 u 和 v 有关的边)，继续这一过程直至 G 中没有边留下。证明这个算法发现一个顶点覆盖，且其大小至多是最小顶点覆盖的两倍。
19. 设计和分析一个哈密顿环问题的穷尽搜索算法。
20. 背包问题(knapsack problem)如下。给定 n 个对象，对象 i 的重量为 w_i，以及一个正整数 K(背包容积)，确定是否存在一个对象的子集它们的整个重量恰为 K。这是一个 NP-完全问题，但有可能设计一个运行时间为 $O(nK)$的算法。这里并没有矛盾，因为 K 可以任意大(例如它可以是 2^n)，因此这个算法是非严格多项式的。用动态程序设计技术设计一个算法。 45

文献提要

有两本书专论串的算法，一本是 Stephen 的著作[180]，另一本是 Crochemore 和 Rytter 的著作[39]。Aho 作了一个出色的回顾[6]。Gusfield[85]的新著与计算生物学密切相关。

不熟悉图理论的读者或许应读一本离散数学教科书中的有关章节。Rosen[163]的书包括了额外的有关图的论著的索引。我们的记号和说明基于 Tarjan 的书的第一章[184]。

两本有关算法的出色的书是 Manber[129]和 Cormen 等人的著作[37]。Manber 强调归纳的使用，而 Cormen 等人的著作则是百科全书式的。Sedgewick[170]，Gonnet 和 Baeza-Yates[76]的书是对各种问题的算法的简要指南。

对 NP-完全问题以及 NP 完全性一般理论的经典论著是 Garey 和 Johnson[68]的书。较新的讨论计算复杂性的是 Papadimitriou[151]的论著。

第 3 章　序列比较与数据库搜索

本章介绍一些最常用的序列比较与数据库搜索方法。序列比较无疑是计算分子生物学中最基本的操作，它直接或间接出现于后续的各章节，在计算机科学的其他领域里也有许多应用。

3.1　生物学背景

47 为什么比较序列？为什么使用计算机进行比较？如何使用计算机进行比较？这些是本章要讨论的主要问题。这一节我们先回答头两个问题，第 3 个将在剩余篇幅讨论。

在计算生物学中序列比较是最重要的原始操作，是许多其他更复杂操作的基础。粗略地讲，这一操作包括发现序列的类同与序列的不同两方面。然而在浅显简单的概念背后存在着各种不同的问题，它们形式也不同，有时需要使用完全不同的数据结构和算法以达到有效的求解。

作为例子，我们接下来给出一组计算生物学问题。在这些例子中我们用到了两个概念，它们将在后面被精确定义。一个概念是两个序列的相似性(similarity)，它给出两个序列相似程度的度量；另一个概念是两个序列的比对(alignment)，它把两个序列叠加后进行对比以揭示相似字符或子串间的对应性。这些例子(以及整个本章中)也用到 2.1 节中定义的有关串的基本概念。计算例如下：

1. 有两个字母表相同、长度相同的串(几万字符)。已知序列几乎是相同的，仅有少
48 数差异，如字符的插入、删除、替换等。这些差异的平均频率很低，几百个字符出现一个。我们需发现差异出现的位置。
2. 有两个字母表相同的序列，各自长数百字符。我们需知道一个序列的前缀是否与一个序列的后缀相似。
3. 问题同 2，但有几百个序列，每一个需与所有其他序列进行对比。而且，绝大部分序列对之间是无关的，即它们缺乏应有的相似度。
4. 有两个字母表相同的序列，各自长数百字符。我们需知道它们之间是否存在相似的子串。
5. 问题同 4，但不是两个序列，而是一个序列与数千个序列比较。

当一个基因被两个不同的实验室测序时，他们需要比较测序的结果，这时将出现类似问题 1 的问题。甚至当一个长序列被输入计算机两次时，我们需寻键入错误，类似情形同样出现。问题 2 与问题 3 出现于大规模 DNA 测序的片段组装过程中。问题 4 与问题 5 出现于大型生物数据库的局部相似性搜索。

本章中我们将看到一个基本的算法概念能够被用于求解所有的上述问题。不过，这可能不是最有效的做法。更快速的算法有时不太普通但更适用于解决具体的问题。

当处理大量的数据时，例如搜索大规模的数据库，使用计算机是不容置疑的。然而，即便我们能够"亲手"进行显而易见的比较，使用计算机也更稳当便捷。Smith 及其同事的工作即是例子[176]。

一篇发表于 1979 年的文章报道了 Sims 及其同事检查两个噬菌体 DNA 的相似性区域[173]。他们比对了噬菌体 St-1 和 G4 的 H 基因的区域，共包含 11 个匹配(两个序列中有 11 列的字符相同)。他们的被认为获得了好比对的方法被描述为"插入少量空隙以形成最大同源性"。而 Smith 及其同事使用计算机却发现了 12 个匹配[176]，因此某些匹配被前一组研究者忽略了。

在另一篇 1979 年的文章中，Rosenberg 和 Court 比对了 46 个取自各种不同生物体的启动子序列[164]。多序列比对从某些区域开始构造，这些区域被认为相对不变，然后将这种"种子"比对向两端扩展，但不引入空隙(gap)。尽管这个多序列比对的总体结果是好的，但据其得到的配对比对却相对较差。这在多序列比对里很常见。但在本例中，Smith 及其同事借助计算机发现了一个两子串间的比对，在总共 45 个字符中有 44 个相同字符，这无疑是个极好的局部比对。而这一事例也未在 Rosenberg 和 Court 的文章中提及，说明 49
他们未能用手工发现这个比对。

这两个例子说明使用计算机能揭示其他方法所不能注意到的隐蔽的相似性。

3.2　比较两个序列

在这节我们研究比较两个序列的方法。我们特别在意找出两序列间的最佳比对。根据我们是关注整个序列的比对还是子串的比对，实践中这一问题有不同的版本，由此导致所谓的全局序列比较和局部序列比较。还存在第三种比较，即不是比较任意子串而是比较给定序列的前缀和后缀，我们称这类比较为半全局比较。所有上述问题均可用动态程序设计方法有效解决。

3.2.1　全局比较——基本算法

考虑如下的 DNA 序列：GACGGATTAG 和 GATCGGAATAG。我们没法不注意到它们看起来十分相似，而当我们把它们作如下对比时，这一事实更加突出：

$$
\begin{array}{l}
\text{G A - C G G A T T A G} \\
\text{G A T C G G A A T A G}
\end{array}
\tag{3.1}
$$

惟一的差别是第二序列中有一个额外 T 以及倒数第四个字母由 T 变成 A。注意，我们必须在第一个序列中引入一个空格，以使得空格前后的碱基完美地对齐。

这一节的目标是介绍一个有效算法确定两序列的最佳比对。当然，讨论问题前我们须定义什么是“最佳”比对。为便于讨论，我们先采取一种简单的形式，后面我们将看到一般形式。

首先，我们先说明什么是两序列间的一个比对。序列可以有不同的大小，比对也可以包含空格。因此，我们把比对定义为在序列中任意位置插入空格(space)使得序列长度相同。等长后，扩展的序列能够完全相互重叠，产生两序列字符间或字符与空格间的对应，但不得有两空格的对应。空格甚至可以插入在序列的头部和尾部。

给定两序列间的一个比对，我们可按如下方法赋予其一个计分。比对的每一列根据其内容获得一个值，比对的总值是各列值之和。如果一列为两个相同的字符，其值为 $+1$(一个匹配)；若为两个不同的字符，其值为 -1(一个失配)；列中含一个空格则值为 -2。最佳比对就是最大计分的比对，这个最大计分称为两序列间的相似性，记做 $\mathrm{sim}(s,t)$，s
50 和 t 是序列。一般地，两序列间可存在多个具有相同最大计分的比对。

为了操练这些定义，我们计算比对(3.1)的计分。它共有 9 个由相同字符组成的列，1 个不同字符组成的列，以及 1 个带空格的列。总的计分是：

$$9\times1+1\times(-1)+1\times(-2)=6$$

为什么我们选择这些特定值$(+1,-1,-2)$? 这个计分系统在实践中经常使用，我们酬奖匹配，减罚失配和空格。在 3.6.2 段，我们将更具体地讨论这些参数的选择。

计算两序列相似性的一个办法是产生所有可能的比对，然后选择最佳的。然而两序列间比对的数量是呈指数增长的，这种办法将导致不可容忍的低效算法。幸好，存在快得多的算法。

这个算法使用一种称做动态程序设计的技术，其特征是利用同一问题已计算出的局部解求解问题本身。给定两个序列 s 和 t，我们的算法不是直接确定 s 和 t 的整体相似性，而是建立在确定两个序列任意前缀的相似性上。我们从最短前缀开始，利用先前的计算结果求解更大前缀的相似性。

令 m 为串 s 的长，n 为串 t 的长。串 s 有 $m+1$ 个前缀，串 t 有 $n+1$ 个前缀(均包括空串)。计算在一个$(m+1)\times(n+1)$的阵列内进行，entry(i,j)包含 $s[1\ldots i]$和 $t[1\ldots j]$的相似性。

图 3.1 显示了 $s=$AAAC 和 $t=$AGC 的阵列，s 排在左列，t 置于顶行，使得前缀结构清晰易见。注意，第一行和第一列初始化为空格罚分(penalty)的积(本例中空格罚分是 -2)，这是因为若一个序列是空的化，两序列间只存在一种可能比对，即在空序列中插入与另一序列字符同样多的空格。这种比对的计分是 $-2k$，k 是非空序列的长度。因此，填写第一行与第一列是容易的。

现在我们观察其他入口项。关键点是，我们能够通过查询三个先前项 entry$(i-1,j)$，entry$(i-1,j-1)$，entry$(i,j-1)$来计算 entry(i,j)的值，其原因是只有恰好三种方式获得 $s[1\ldots i]$和 $t[1\ldots j]$之间的比对，每一种使用这三个先前项之一。实际上，要
51 得到 $s[1\ldots i]$和 $t[1\ldots j]$之间的比对，我们有如下三种选择：

- 比对 $s[1\ldots i]$与 $t[1\ldots j-1]$，在 $t[j]$处匹配一个空格，或
- 比对 $s[1\ldots i-1]$与 $t[1\ldots j-1]$，$s[i]$与 $t[j]$匹配，或

- 比对 $s[1...i-1]$与 $t[1...j]$,在 $s[i]$处匹配一个空格。

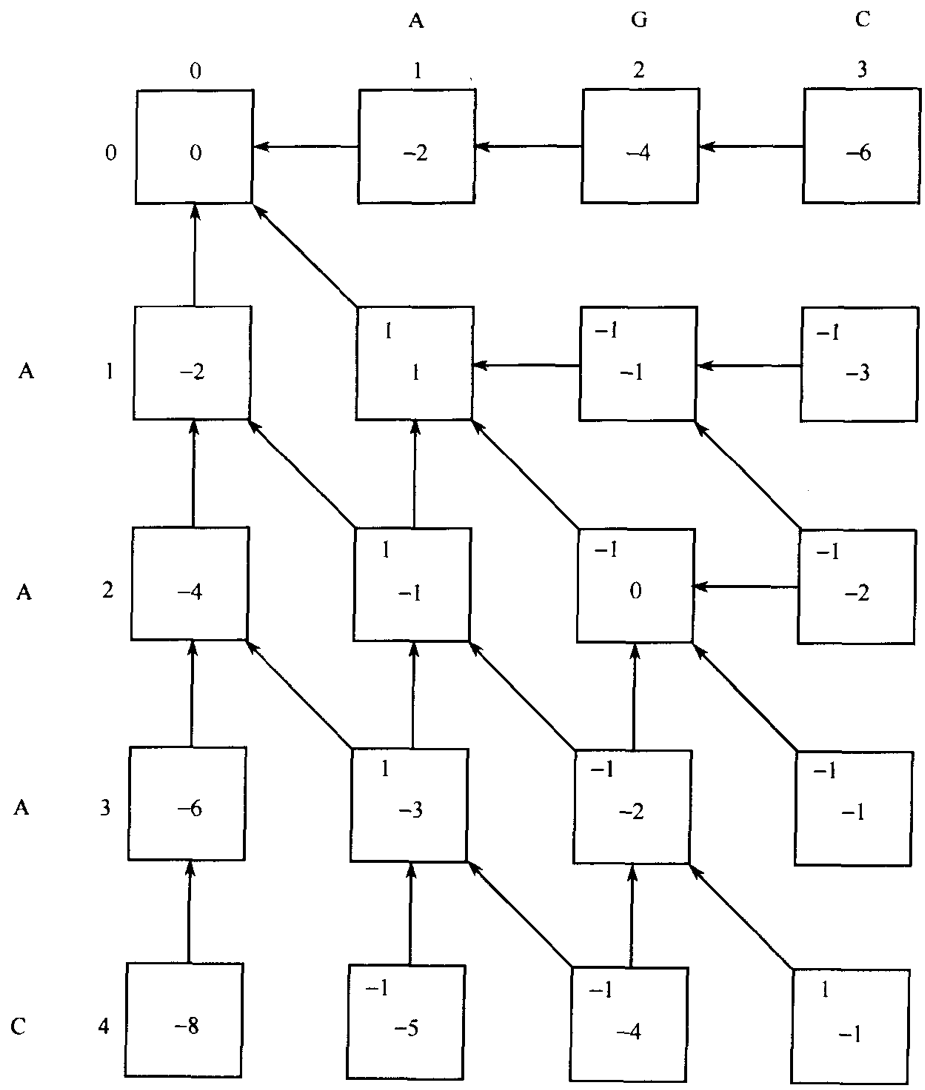


图 3.1　计算优化比对的二维阵列。单元(i,j)左上角的值表示是否有 $s[i]=t[j]$。行和列的下标由 0 开始

这些可能性是穷尽的,因为在最后一列我们不能以两个空格相互匹配。如果采用合适的次序计算各入口项,则每当计算一个入口项时相关各子前缀的最佳比对的计分已存储于阵列中,这样,各个前缀的相似性可按下列公式确定 52

$$\mathrm{sim}(s[1...i],t[1...j])=\max\begin{cases}\mathrm{sim}(s[1...i],t[1...j-1])-2\\ \mathrm{sim}(s[1...i-1],t[1...j-1])+p(i,j)\\ \mathrm{sim}(s[1...i-1],t[1...j])-2\end{cases}\tag{3.2}$$

这里,如果 $s[i]=t[j]$则 $p(i,j)$等于 +1,否则如果 $s[i]\neq t[j]$,$p(i,j)$则等于 −1。$p(i,j)$的值写在图 3.1 中各入口项框的左上角。如果称阵列为 a,这个公式可写成如下形式:

$$a[i,j]=\max\begin{cases}a[i,j-1]-2\\a[i-1,j-1]+p(i,j)\\a[i-1,j]-2\end{cases}\tag{3.3}$$

如前所述，必须使用一个好的计算次序，而这很容易做到，只要从左到右逐行填写阵列或从上到下逐列填写阵列即可。任何其他次序，只要在计算 $a[i,j]$ 之前可得到 $a[i,j-1]$，$a[i-1,j-1]$，$a[i-1,j]$ 的值，也均有效。

最后，我们在图 3.1 中画出箭头，根据公式(3.3)指向最大值来自的方向。例如，$a[1,2]$ 的值取自如下三项的最大者

$$a[1,1]-2=-1$$
$$a[0,1]-1=-3$$
$$a[0,2]-2=-6$$

因此，只有一条途径得到这个最大值，即来自 $a[1,1]$，这就是箭头所指的方向。

图 3.2 是填写阵列的算法，该算法逐行计算入口项。它依赖一个参数 g 来确定空格罚数(通常 $g<0$)和一个计分函数 p 来计算字符的配对。我们曾经在图 3.1 中令 $g=-2$，如果 $a=b$ 则 $p(a,b)=1$，如果 $a\neq b$ 则 $p(a,b)=-1$。

```
Algorithm Similarity
    input: sequences s and t
    output: similarity between s and t
    m ← |s|
    n ← |t|
    for i ← 0 to m do
        a[i, 0] ← i × g
    for j ← 0 to n do
        a[0, j] ← j × g
    for i ← 1 to m do
        for j ← 1 to n do
            a[i, j] ← max(a[i − 1, j] + g,
                          a[i − 1, j − 1] + p(i, j),
                          a[i, j − 1] + g)
    return a[m, n]
```

53 图 3.2 两序列比较的基本动态程序设计算法

= = = = = = = = = = = 优 化 比 对 = = = = = = = = = = =

我们已讨论了如何计算两序列间的相似性。这里，使用图 3.1 中的箭头，我们将考察如何构造序列间的优化比对。我们需要做的是从 entry(m,n)开始，循箭头移动，直至 entry(0,0)，每个箭头将告诉我们比对的一列。考察离开 entry(i,j)的箭头，如果这个箭头是水平的，它对应于 s 中一带有一个空格的列与 $t[j]$匹配；如果箭头是垂直的，它对应于 $s[i]$匹配于 t 中一空格；斜箭头表征 $s[i]$与 $t[j]$匹配。注意第一个序列 s 总是垂直排列于左侧。因此，如果我们已用基本算法事先计算出了阵列 a，则优化比对可以容易地从右到左构造出来，无需显式地引入箭头——使用一个简单测试即可判断下一个要访问的入口项。

图 3.3 给出了一个递归算法用于确定给定序列 s、t 和阵列 a 的一个优化比对。调用 *Align*(m , n , *len*)将构造一个优化比对,答案由一对向量 *align-s* 和 *align-t* 给出,在长度 1…*len* 里列出比对的字符,这些字符可以是空格或字母。向量 *align-s* 和 *align-t* 当做全局变量处理,比对的长度由算法通过参数 *len* 返回。注意,$\max(|s|,|t|)\leqslant len\leqslant m+n$。

```
Algorithm Align
    input: indices i, j, array a given by algorithm Similarity
    output: alignment in align-s, align-t, and length in len
    if i = 0 and j = 0 then
        len ← 0
    else if i > 0 and a[i, j] = a[i − 1, j] + g then
        Align(i − 1, j, len)
        len ← len + 1
        align-s[len] ← s[i]
        align-t[len] ← -
    else if i > 0 and j > 0 and a[i, j] = a[i − 1, j − 1] + p(i, j) then
        Align(i − 1, j − 1, len)
        len ← len + 1
        align-s[len] ← s[i]
        align-t[len] ← t[j]
    else    // has to be j > 0 and a[i, j] = a[i, j − 1] + g
        Align(i, j − 1, len)
        len ← len + 1
        align-s[len] ← -
        align-t[len] ← t[j]
```

图 3.3　优化比对的递归算法 54

如前所述,对给定的序列对可以存在许多优化比对。图 3.3 的算法仅返回其中一个比对值,在离开 entry(i , j)时以逆时针方向优先选取边(见图 3.4),结果是由这种算法给出的优化比对具有如下特性:当存在选择时,t 中带空格的列优先于两个字母列,后者又优先于 s 中带空格的列。例如,当比对 s = ATAT, t = TATA 时,我们得到[注意是从 entry(m , n)开始]

－ A T A T
T A T A －

而不是

A T A T －
－ T A T A

后者是这两个序列的另一个优化比对。类似地,当比对序列 s = AA 与序列 t = AAAA 时,我们得到

－ － A A
A A A A

尽管这时存在 5 种其他优化比对。这种比对有时也称做"最上"比对,因为它使用阵列中的上方箭头。反之,若反向改变算法中 if 语句的次序我们将得到"最下"比对。当某列既出现于最上比对又出现于最下比对时,该列将出现于所有优化比对中。

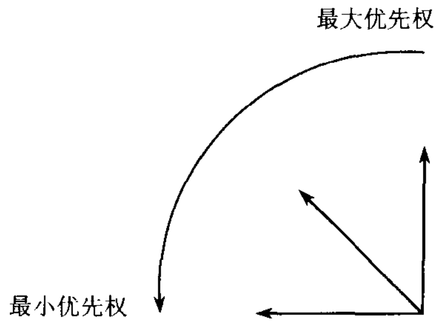


图 3.4 箭头优先方向

可以修改算法使之产生两序列间的所有优化比对,这时我们需要使用一个带指针的栈,指针指向候选项和回溯项,以便穷尽沿箭头由 entry(i,j)抵达 entry(0,0)的所有可能途径。不过,优化比对的数量可能非常大。在这样的例子里我们常建议只保留全部或部分比对的某种简短的表示,例如,最上与最下比对。

现在让我们确定算法的复杂性。图 3.2 中的基本算法有 4 个循环,前两个做初始化,分别消耗时间 $O(m)$和 $O(n)$,后两个循环是嵌套的,填写阵列的其余元素。操作的数量本质上取决于入口项的数量(即阵列的大小)。因此算法在这部分花费时间为 $O(mn)$,并且这是时间复杂性的支配项。所需空间也与阵列大小成比例。因此,基本算法的时间与空间复杂性均是 $O(mn)$。如果序列有相似或相等的长度,例如 n,复杂性便是 $O(n^2)$,这就是我们说这些算法具有平方复杂性的原因。

在阵列已填充的情况下,比对的复杂性是 $O(len)$,len 是所返回比对的长度,它等于 $O(m+n)$。

3.2.2 局部比较

序列 s 和 t 之间的局部比对(local alignment)是其子串间的比对。在这部分我们介绍一个找出两序列局部比对最高计分的算法。

这个算法是基本算法的一个变体,主要数据结构仍是$(m+1)\times(n+1)$ 阵列,只是阵列值的解释不同。每个入口项 entry(i,j)将存记 $s[1\ldots i]$的后缀和 $t[1\ldots j]$的后缀之间比对的最高计分。第一行和第一列初始化为 0。

对任何入口项 entry(i,j),一定存在一个计分为 0 的比对,而这个比对是由 $s[1\ldots i]$的空后缀和 $t[1\ldots j]$的空后缀所形成。因此,阵列的所有入口项都大于或等于 0,这部分地解释了以上的初始化。

初始化之后,阵列将按通常方式填充,$a[i,j]$的值取决于 3 个先前计算的入口项,形成如下的递归定义

$$a[i,j]=\max\begin{cases}a[i,j-1]+g\\ a[i-1,j-1]+p(i,j)\\ a[i-1,j]+g\\ 0\end{cases}$$

除了这里的第四个选项没有在全局比较中出现,其余均与基本算法一样。

最后,我们将在整个阵列中找最大入口项,它包含了优化局部比对的计分。任何包含这个值的入口项均可作为比对的起始点,比对的其余步骤可按通常的回溯进行,一旦遇到无输出箭头的入口项即告终止。或者,我们也可终止于记分值为 0 的入口项。

一般来说,做局部比对时我们不仅关心优化比对也关心计分超过某阈值的近似优化比对。有关寻找近似优化比对的方法可见文献提要。 56

3.2.3 半全局比较

在半全局比较里,我们在作计分计算时忽略序列终端的空格。而基本动态程序设计算法的一个有趣特征是,人们能够通过对原有机制稍作修改而控制终端空格的罚数。

让我们从精确定义终端空格开始,并说明为什么在某些情况下适宜让它们自由添加。终端空格是那些出现于序列头字符之前或尾字符之后的空格。例如,下面的比对中出现于第二序列的所有空格均是终端空格,而第一序列中的空格则不是终端空格:

C A G C A - C T T G G A T T C T C G G
- - - C A G C G T G G - - - - - - - -　(3.4)

注意到这两个序列的长度差异较大,一个有 8 个字符,而另一个有 18 个字符。当这种情况发生时,任何一种比对都将包含大量空格,对计分贡献一个大的负值。然而,如果我们忽略终端空格则这个比对相当漂亮,6 个匹配,1 个失配,1 个空格。

通过观察可发现这尚不是这两序列间的最佳比对。在下面的比对(3.5)中,根据我们迄今所用的计分系统,我们呈示另一个更高计分的比对(相对于上例的 - 19,本例是 - 12):

C A G C A C T T G G A T T C T C G G
C A G C - - - - - G - T - - - - G G　(3.5)

尽管有更高的计分且第二个串全部字符均达到匹配,从发现序列间相似区域来讲这个比对并不那么有趣。仅仅为了字符间的完全匹配,第二个序列被空格强制性地打散。如果我们要寻找长序列中与短序列相似的区域,无疑第一种比对更切合目标。如果我们忽略终端空格,比对(3.4)获得 3 分,而比对(3.5)仍旧是 - 12 分。

现在我们介绍基本算法的一个忽略终端空格的变体。首先我们考虑这样的情形,即串 s 最后字符后的空格不计分,作一个优化比对,s 终端后的空格匹配到 t 的后缀。如果我们移去比对的最后部分,剩余部分是串 s 和串 t 的前缀间的一个比对,其计分与原始情形的比对相同。因此,要得到 s 和 t 的不计终端空格的计分我们所需做的就是找出 s 和 t 的前缀间的最佳比对。但我们在 3.2.1 段中看到阵列 a 的 entry(i,j)中包含了 $s[1\ldots i]$ 和 $t[1\cdots j]$的相似性,因此只要在 a 的最后一行取最大值即可,即

$$\mathrm{sim}(s,t)=\max_{j}^{n} a[m,j]$$

注意,这里我们改变了 $\mathrm{sim}(s,t)$的定义,使得它表征相似性并忽略 s 中最后的空格。这个表达式给出了最佳比对的计分。要找出比对本身,我们只需按基本算法行事,但是从 57
(m,k)开始,k 应满足 $\mathrm{sim}(s,t)=a[m,k]$。

一个类似的做法是忽略 t 中的终端空格,此时,我们在 a 的最后一列取到比对的最

大计分。我们甚至可以综合两种情况寻找最佳比对,忽略任一序列中的终端空格。最大计分可在阵列的最后一行和最后一列构成的边界中找到。在各种例子里,要找出一个优化比对,我们可从包含相似性值的一个入口项开始,循箭头前进搜索至 entry(0,0)。如同在基本例中一样,每个箭头给出比对中的一列。

现在我们转到序列带初始空格的情形。假定我们需要不计 s 中初始空格的最佳比对,这等价于 s 与 t 的一个后缀的最佳比对。为得到所需答案,我们使用一个与基本算法类似但稍有差别的 $(m+1)\times(n+1)$ 阵列,每个入口项 entry(i,j) 包含 $s[1...i]$ 和 t 的一个前缀的后缀(即 $t[1...j]$ 的后缀)之间的最高记分。

显而易见,$a[m,n]$ 即是答案。不那么明显但很确定的是,这个阵列能用与基本算法相同的公式进行计算填充,即公式(3.3)。不过,初始化有所不同。由于入口项的新意义,第一行将用 0 而不是空格罚数的积来初始化。我们把检验公式(3.3)对这一情形确实有效这一问题留给读者。

我们可应用同样的技巧用 0 初始化第一列,并忽略 t 的起始空格。此外,如果我们用 0 同时初始化第一行和第一列,并用公式(3.3)计算其他的入口项,我们将得到 s 的后缀和 t 的后缀间的最大相似性。要找出最佳比对,我们须从最大值入口项出发,循箭头搜索,直至达到任一初始化为 0 的边界入口项,然后循边界至原点。

表 3.1　终端空格罚分过程概述

空格不记分的位置	动　作
第一序列的起始	用 0 初始化第一行
第一序列的终端	在最后一行寻找最大值
第二序列的起始	用 0 初始化第一列
第二序列的终端	在最后一列寻找最大值

表 3.1 概述了这几种变体。存在 4 种不对空格计分的情形:s 的起始和终结,以及 t 的起始和终结。可以按任意方式独立地组合这些条件并使用基本算法的变体找出两序列相似性。惟一改变的是初始化以及在哪里寻找最大值。忽略初始空格可转换为用 0 初始化某些位置,而忽略终端空格意味着从某些特定位置开始寻找最大值。
58 填充阵列的过程总是相同的,即使用公式(3.3)。

3.3　基本算法的扩展

3.2 节中描述的算法对大多数应用都是适用的。但有时我们会遇到特殊情形,需要更好的算法。在这部分我们研究几种能够用于这些特殊情形的技术。

算法的第一类改进与问题的计算复杂性有关。我们将证明,在大致将计算时间增加一倍的代价下可以把算法的空间需求从平方性的(mn)减少到线性的$(m+n)$。另一方面,我们也可以减少算法的时间复杂性,但这种情况只适用于相似的序列以及某些计分参数。

算法的另一类改进涉及比对的生物学解释。从生物学的角度看,一系列连续的空格要比独立的空格更具现实性。我们将研究符合这种看法的基本算法的变体。

3.3.1　空间节省

基本算法的平方复杂性使它不便于比较非常长的序列以及反复比较多个序列。尽管

对参数做出限制时确实存在更快的算法，但费时少而又保持一般性的算法是不存在的。

对于空间，则有可能在保持算法一般性的条件下降低算法的复杂性，所付的代价是计算时间大致翻倍。然而，当渐进时间复杂性不变时，对于多数应用而言限制因素是空间而非时间。因此，空间复杂性的改进特别有实际意义。这部分我们描述一个漂亮的空间节省技术。

首先，我们注意到 sim(s,t)的计算可在线性空间里完成。由于阵列的每一行仅仅依赖于前一行，计算中仅需在内存中保留一个向量，它由部分当前计算行和部分先前计算行构成，见图 3.5 中的代码。显然，对列的情况也如此，且如果 $m<n$，对列使用这个技巧还可节约空间。图 3.5 中，在 i 循环每次迭代的终末，向量 a 包含了 $s[1\ldots i]$与 t 的所有前缀间的相似性。这一事实将在以后用到。 59

```
Algorithm BestScore
    input: sequences s and t
    output: vector a
    m ← |s|
    n ← |t|
    for j ← 0 to n do
        a[j] ← j × g
    for i ← 1 to m do
        old ← a[0]
        a[0] ← i × g
        for j ← 1 to n do
            temp ← a[j]
            a[j] ← max(a[j] + g,
                       old + p(i, j),
                       a[j − 1] + g)
            old ← temp
```

图 3.5　线性空间中相似性的算法。最终，$a[n]$包含 sim(s,t)

算法中困难的部分是要在线性空间里得到优化比对。在我们先前所见的算法里，完成优化比对需要整个阵列。为了克服这个困难，我们使用“分而治之”的策略，即把问题分解为两个子问题，然后再组合子问题的解以得到整个问题的解。

关键的思想是这样的。固定一个优化比对以及 s 中的一个位置 i，然后考虑 $s[i]$在这个比对中能匹配到什么。不外乎有如下两种可能性：

(1) 对 $1\ldots n$ 中的某个j，字母 $t[j]$匹配 $s[i]$

(2) 对 $0\ldots n$ 中的某个j，$t[j]$和 $t[j+1]$之间的空格匹配 $s[i]$

在第二种情况中，下标 j 在 0 到 n 之间变化，因为在一个序列中容纳空格的位置总比容纳字母的位置多一个。我们没有区分 $j=0$ 或 $j=n$，即空格在 $t[1]$之前或 $t[n]$之后。

令

$$Optimal\binom{x}{y}$$

表示 x 与 y 之间的一个优化比对。s 与 t 之间的每一个比对，不论是否是优化的，均满足

(1)或(2)。特别地,我们固定的优化比对也必须满足这两者之一。如果它满足(1),为了得到完整的比对,我们必须拼接

$$Optimal\begin{pmatrix} s[1\ldots i-1] \\ t[1\ldots j-1] \end{pmatrix} + \begin{matrix} s[i] \\ t[j] \end{matrix} + Optimal\begin{pmatrix} s[i+1\ldots m] \\ t[j+1\ldots n] \end{pmatrix} \tag{3.6}$$

如果它满足(2),我们则必须拼接

$$Optimal\begin{pmatrix} s[1\ldots i-1] \\ t[1\ldots j] \end{pmatrix} + \begin{matrix} s[i] \\ - \end{matrix} + Optimal\begin{pmatrix} s[i+1\ldots m] \\ t[j+1\ldots n] \end{pmatrix} \tag{3.7}$$

这些揭示了一个计算优化比对的递归方法,其前提是如果对给定的 i 我们能够确定(1)与(2)中的哪一个将会出现以及 j 的对应值。

处理方法如下。根据方程(3.6)和(3.7),对固定的 i,我们需要 $s[1\ldots i-1]$和 t 的任
60 意前缀之间的相似性,以及 $s[i+1\ldots m]$和 t 的任意后缀之间的相似性。一旦有了这些值,我们就能计算(3.6)中所描述的 j 比对的计分和(3.7)中所描述的 $j+1$ 比对的计分。通过选择最佳者,我们就有了进行递归处理所需的信息。

如前所述,我们能在线性空间里计算 s 的一个给定前缀与 t 的所有前缀间的最优计分(图 3.5)。对后缀也存在类似算法。至此我们的问题已接近解决,尚存的惟一问题是每次递归调用中决定 i 取什么值,最佳的选择是令 i 尽可能接近序列的中间。完整的代码在图 3.6 中给出。在这个算法中,调用

$$BestScore(s[a\ldots i-1], t[c\ldots d], pref\text{-}sim)$$

用 *pref-sim* 返回 $s[a\ldots i-1]$和 $t[c,j]$之间的相似性,j 取自 $c-1\ldots d$。类似地,调用

$$BestScoreRev(s[i+1\ldots b], t[c\ldots d], suff\text{-}sim)$$

用 *suff-sim* 返回 $s[i+1\ldots b]$和 $t[j+1\ldots d]$之间的相似性,j 取自 $c-1\ldots d$。调用

$$Align(1, m, 1, n, 1, len)$$

将返回一个优化比对,存于全局变量 *align-s* 和 *align-t*,比对的长度存于 *len* 中。

最后需要考虑的一个问题是,由于这些额外的计算,计算时间是否显著上升?情况尚好,下面的分析表明时间只是大致翻倍。

当 $m=b-a+1, n=c-d+1$ 时,调用 $Align(a,b,c,d,start,end)$,在 *BestScore* 或 *BestScoreRev* 的最内层计算最大值(最优计分),$T(m,n)$是计算最大值的次数。容易看出整个处理时间与 $T(m,n)$成比例,加上控制与初始化所需的线性项时间,我们可证明 $T(m,n)\leqslant 2mn$。

通过对 m 进行归纳可得到一个证明。若 $m=1$,无最大值计算,因此显然有 $T(1,n)\leqslant 2n$。若 $m>1$,我们将调用 *BestScore*,至多有 $mn/2$ 次最大值计算,且对 *BestScoreRev* 有同样的调用和同样次数的最大值计算,对 *Align* 的两次递归调用分别产生至多 $T(m/2,j)$和 $T(m/2,n-j)$次最大值计算。把这些计算次数相加得到:

$$\begin{aligned} T(m,n) &\leqslant \frac{mn}{2} + \frac{mn}{2} + T(m/2,j) + T(m/2,n-j) \\ &\leqslant mn + mj + m(n-j) \end{aligned}$$

61 $$= 2mn$$

```
Algorithm Align
    input: sequences s and t, indices a, b, c, d, start position start
    output: optimal alignment between s[a..b] and t[c..d] placed in vectors align-s
        and align-t beginning at position start and ending at end
    if s[a..b] empty or t[c..d] empty then
        // Base case: s[a..b] empty or t[c..d] empty
        Align the nonempty sequence with spaces
        end ← start + max(|s|, |t|)
    else
        // General case
        i ← ⌊(a + b)/2⌋
        BestScore(s[a..(i − 1)], t[c..d], pref-sim)
        BestScoreRev(s[(i + 1)..b], t[c..d], suff-sim)

        posmax ← c − 1
        typemax ← SPACE
        vmax ← pref-sim[c − 1] + g + suff-sim[c − 1]
        for j ← c to d do
            if pref-sim[j − 1] + p(i, j) + suff-sim[j] > vmax then
                posmax ← j
                typemax ← SYMBOL
                vmax ← pref-sim[j − 1] + p(i, j) + suff-sim[j]

            if pref-sim[j] + g + suff-sim[j] > vmax then
                posmax ← j
                typemax ← SPACE
                vmax ← pref-sim[j] + g + suff-sim[j]

        if typemax = SPACE then
            Align(a, i − 1, c, posmax, start, middle)
            align-s[middle] ← s[i]
            align-t[middle] ← SPACE
            Align(i + 1, b, posmax + 1, d, middle + 1, end)
        else // typemax = SYMBOL
            Align(a, i − 1, c, posmax − 1, start, middle)
            align-s[middle] ← s[i]
            align-t[middle] ← t[posmax]
            Align(i + 1, b, posmax + 1, d, middle + 1, end)
```

图 3.6　使用线性空间的一个优化比对算法

3.3.2　一般空隙罚分函数

我们首先把空隙定义为数目 $k>1$ 的空格串。通常认为当突变发生时由 k 个空格组成的空隙的出现率高于 k 个孤立空格的出现率，这是因为空隙可由移去一整段碱基的单个突变事件产生，而分离的一连串空格则是由不同的一串事件产生的，单个事件通常要比一串事件常见。

迄今为止我们尚未对空格簇与孤立空格加以区分，这意味着一个空隙是通过一个线性函数来减罚的。一个由 k 个空格构成的空隙的减罚函数定义为 $w(k)$，$k\geqslant 1$，且 62

$$w(k)=bk$$

这里 b 是一个与空格的计分有关的绝对值。

在本部分，我们介绍一个针对一般空隙减罚函数 w 的计算相似性的算法。假定序列长度为 n，这个算法具有时间复杂性 $O(n^3)$，因此慢于基本算法。但它与基本算法有许多共同点。主要差别是计分不是可加性(additive)的，因为我们不能把一个比对拆成两部分并把两部分计分的和作为比对计分。特别是，我们不能把比对的最后一列分离出来且把最后一列的计分与其余部分的计分之和作为比对计分。不过，如果我们把比对按块边界拆分，则计分的累加性仍然有效。每一个比对能够被惟一性地分解为一系列连续的块，且存在如下三类块：

1. Σ中两个已比对的字符
2. t 中一个最大连续字符序列与 s 中空格串的比对
3. s 中一个最大连续字符序列与 t 中空格串的比对

"最大"意味着它不能够再扩展。图 3.7 显示了一个比对以及它的分块。上述第一类中的块产生计分 $p(a,b)$，a 和 b 是两个比对的序列，第二和第三类的块产生计分 $-w(k)$，k 是空隙的长度。

```
s1    AAC---AATTCCGACTAC
s2    ACTACCT------CGC--
```

s_1	A	A	C	---	A	TATCCG	A	C	T	AC
s_2	A	C	T	ACC	T	------	C	G	C	--

图 3.7 一个比对及其块

比对的计分现在不是在列的层次进行计算，而是在块的层次进行计算。相应地，仅当按块边界拆分时我们才能保持计分的累加性。这意味着动态程序设计算法要作一些显著的改变以进行一般空隙减罚函数下的相似性计算。我们不从比对的最后一列进行推延，而是须从比对的最后一个块进行推延。而且，块之间不能随意排列，一个类型 2 或类型 3 的块不能跟在另一个同类型块之后。对于每一对(i,j)，这要求我们不是保留前缀 $s[1...i]$ 和 $t[1...j]$之间比对的最优计分，而是要保留终止于某个特定类型块之前的这些前缀的最优计分。

为了比较长为 m 的串 s 和长为 n 的串 t，我们使用 3 个大小为$(m+1)\times(n+1)$的数组，每个终止块一个数组。数组 a 用于以字符—字符终止的块的比对，b 用于以 s 中空格终止的块的比对，c 用于以 t 中空格终止的块的比对。根据数组的意义，第一行和第一列
63 的初始化如下所示：

$$a[0,0]=0$$
$$b[0,j]=-w(j)$$
$$c[i,0]=-w(i)$$

初始行与列中的所有其他位置填入 $-\infty$，以使它们对最大值的计算无影响。

我们现在要问，什么类型的块终止优化比对？这个问题将决定 a,b,c 哪个数组将进

行更新。它们之间的递归关系如下：

$$a[i,j]=p[i,j]+\max\begin{cases}a[i-1,j-1]\\b[i-1,j-1]\\c[i-1,j-1]\end{cases}$$

$$b[i,j]=\max\begin{cases}a[i,j-k]-w(k), for\ 1\leqslant k\leqslant j\\c[i,j-k]-w(k), for\ 1\leqslant k\leqslant j\end{cases}$$

$$c[i,j]=\max\begin{cases}a[i-k,j]-w(k), for\ 1\leqslant k\leqslant i\\b[i-k,j]-w(k), for\ 1\leqslant k\leqslant i\end{cases}$$

通常，$p(i,j)$表征 $s[i]$与 $t[j]$之间一个匹配的计分。

注意，数组 b 和 c 中的入口项依赖于一个以上的先前值，因为最后一个块可有不定的长度。而且，当计算 b 中一个入口项时我们不查看它的前面的 b 入口项，因为一个类型 2 的块不能紧接着另一个类型 2 的块；同样，c 的入口项也不直接依赖于其前面的 c 入口项。为了获得最终的答案即 $\mathrm{sim}(s,t)$的值，我们取 $a[m,n]$，$b[m,n]$，$c[m,n]$的最大值。

该算法的时间复杂性是 $O(mn^2+m^2n)$。为了看出这一点，根据给定的公式，计数某些数组中某些入口项的读取次数，这是时间复杂性中的关键项。容易看出，为了计算 $a[i,j]$，$b[i,j]$，$c[i,j]$，我们需要访问先前的数组入口项

$$3+2j+2i$$

次，故总计需访问整个数组

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(3+2j+2i)$$

次。上面的和式能用如下的近似算式计算：

$$\sum_{j=1}^{n}(2i+2j+3)=2ni+n(n+1)+3n=2ni+n^2+4n$$

$$\sum_{i=1}^{m}(2ni+n^2+4n)=nm(m+1)+mn^2+4mn=m^2n+5mn+mn^2$$

获得优化比对是直截了当的，上面算法中的概念仍旧适用。在数组中我们沿着对计算最大值有贡献的入口项回溯，并记住所用到的数组。 64

由此我们看出，任何减罚函数都是可实现的，所付出的代价是可观的时间与空间开销。当基于动态程序设计的序列比较成为更大计算任务中的瓶颈因素时，这种额外开销可能是很关键的。

3.3.3　仿射空隙罚分函数

我们已知当序列长度为 n 时，带线性减罚函数的基本算法的时间复杂性是 $O(n^2)$，我们还知道引入一般减罚函数将使运行时间增加到 $O(n^3)$。现在的问题是，如果我们使用一个不那么一般化的减罚函数，但它仍然使得由 k 个空格组成的空隙产生的罚数小于 k 个孤立空格产生的罚数，我们是否仍能得到 $O(n^2)$的算法？在这一部分我们将说明这是可能的。

如果我们要引入这样的概念，即 k 个连续的空格要比 k 个孤立的空格更可能出现，我们必须有

$$w(k) \leqslant kw(1)$$

或，更一般地

$$w(k_1 + k_2 + \cdots + k_n) \leqslant w(k_1) + w(k_2) + \cdots + w(k_n) \tag{3.8}$$

一个满足式(3.8)的函数 w 称次加性函数(subadditive function)。如果它满足 $w(k) = h + gk, k \geqslant 1$，且 $w(0) = 0$，则函数 w 是一个仿射函数(affine function)；如果 $h, g > 0$，则它是次加性的。

换一个角度认识函数 w，我们可以说空隙中的第一个空格的罚值是 $h + g$，而其他空格则是 g。这种耗费源自从空格 $k-1$ 到空格 k 的减罚的差 $\Delta w(k) = w(k) - w(k-1)$。

我们现在描述这种情形下的动态程序设计算法，其与基本算法的主要差别是：这里我们需要区别空隙中的第一个空格和其他空格以进行不同的减罚。这是通过使用 3 个数组来达到的，如同在一般空隙减罚函数的例子中那样。每一个数组中的入口项具有如下意义：

$a[i,j] = s[1...i]$ 与 $t[1...j]$之间比对的最大计分，$s[i]$与 $t[j]$匹配。

$b[i,j] = s[1...i]$与 $t[1...j]$之间比对的最大计分，s 的一个空格与$t[j]$匹配。

$c[i,j] = s[1...i]$与 $t[1...j]$之间比对的最大计分，$s[i]$与 t 的一个空格匹配。

根据如下公式，这些数组的入口项依赖于先前的入口项：

65

$$a[i,j] = p(i,j) + \max\begin{cases} a[i-1,j-1] \\ b[i-1,j-1] \\ c[i-1,j-1] \end{cases}$$

$$b[i,j] = \max\begin{cases} -(h+g) + a[i,j-1] \\ -g + b[i,j-1] \\ -(h+g) + c[i,j-1] \end{cases}$$

$$c[i,j] = \max\begin{cases} -(h+g) + a[i-1,j] \\ -(h+g) + b[i-1,j] \\ -g + c[i-1,j] \end{cases}$$

i, j 的范围是 $1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n$。与前面一样，$p(i,j)$表征 $s[i]$与 $t[j]$之间的匹配。

现在我们来理解一下这个公式。$a[i,j]$的公式包含 $p(i,j)$项，它是最后一列的计分，加上前缀 $s[1...i-1]$与 $t[1...j-1]$之间比对的最佳计分。

我们知道 $b[i,j]$意味着 s 的最后一列是一个空格，我们仍须检查它是空隙的起始空格还是空隙的后续空格以便正确地减罚。首先，我们要从数组读进相关前缀 $s[1...i]$和 $t[1...j-1]$的最佳计分，这正是我们在 $b[i,j]$的公式中留意入口项$(i,j-1)$的原因。对应于 $a[i,j-1]$和 $c[i,j-1]$的比对不以 s 中的空格结尾，因此 s 中的最后空格$s[i]$必须作为一个空隙的起始空格进行减罚，减去 $h+g$。对应于 $b[i,j-1]$的比对在 s 的最后一列已经是空格，因此对应于 $t[j]$的空格(即 $s[i]$)应是空隙的后续空格，只需减去 g。这三个可能性已穷尽了各种情况，因此其中的最大值就是正确值。对于 $c[i,j]$情形类同。

这些数组的初始化需要加以注意，我们再次说明，它取决于我们是否对起始空格进行

减罚。我们这里假定对所有空格进行减罚，在其他情形下对算法的修改留给读者。

需要初始化的入口项是下标为$(i,0)$的项和下标为$(0,j)$的项，$0\leqslant i\leqslant m$，$0\leqslant j\leqslant n$。初始化须按定义中的每个矩阵的内容确定。因此，对矩阵 b，我们有

$$b[i,0]=-\infty, 0\leqslant i\leqslant m$$
$$b[0,j]=-(h+gj), 0\leqslant j\leqslant n$$

把 $-\infty$ 赋值给 $b[i,0]$是为了与 $b[i,j]$的定义相符，因为在 $s[1\cdots i]$与 $t[1\cdots 0]$(空序列)之间没有比对，故也不存在 s 中的一个空格与 $t[0]$匹配(根本就没有 $t[0]$)。如果没有比对，最大值须是 $-\infty$，这也是最大记分值。另一方面，在 $s[1\cdots 0]$(空序列)与 $t[1\cdots j]$之间，恰存在一个以 s 中空格结尾的匹配，这个比对有计分 $-(h+gj)$。

以类似的方式，我们初始化 c 为

$$c[i,0]=-(h+gi),\qquad 1\leqslant i\leqslant m$$
$$c[0,j]=-\infty,\qquad 1\leqslant j\leqslant n$$

以及最后

$$a[0,0]=0$$
$$a[i,0]=-\infty,\qquad 1\leqslant i\leqslant m$$
$$a[0,j]=-\infty,\qquad 1\leqslant j\leqslant n$$ 66

在 a 的初始化中，我们用到了不以空格终止的比对，推广了给定的定义。为得到最后的结果，即相似性，我们取 $a[m,n]$，$b[m,n]$，$c[m,n]$的最大值即可，因为这已涵盖了所有的可能性。

类似于针对一般空隙罚分函数构造优化比对的例子，我们这里也能构造一个优化比对。从最后的入口项(m,n)起，沿着在填写数组过程中曾经选作最大值的入口项进行回溯，直至达到起始点$(0,0)$。在回溯中，不仅要记住入口项的当前位置，还要记住它属于哪个数组(a，b，或 c)。

这个算法的时间复杂性确实是 $O(mn)$，我们把这一点留给读者检验。空间复杂性也是 $O(mn)$，但存在线性空间复杂性的版本(见文献提要)。

3.3.4　比较相似序列

当两个序列在某种程度上“看起来”相像时它们就是相似的。我们有了直观的相似性概念，并通过比对和计分对它进行了形式化。因此，对我们来说，当两个序列之间的优化比对计分非常接近最大可能值时(我们将精确定义“最大可能”)，它们便是相似的。这部分讨论在这种情况下找出好比对的快速算法。我们仅分析全局比对。

我们首先考虑两个序列 s 和 t 具有同样长度 n 的情形。这是一个合理的假设，因为我们关注的是相似序列。如果 s 和 t 具有同样长度，动态程序设计矩阵是一个方阵，它的主对角线从$(0,0)$到(n,n)。这条对角线对应着 s 与 t 间惟一的不带空格的比对。如果这不是一个优化比对，我们需要在序列中插入一些空格以获得更佳的计分。空格总是成对加入，s 中加一个 t 中也加一个。

随着我们加入空格对，比对离开主对角线。考虑如下两个序列

$$s = \mathrm{G\ C\ G\ C\ A\ T\ G\ G\ A\ T\ T\ G\ A\ G\ C\ G\ A}$$
$$t = \mathrm{T\ G\ C\ G\ C\ C\ A\ T\ G\ G\ A\ T\ G\ A\ G\ C\ A}$$

优化比对对应于那些接近主对角线两次但不在其上的通路，见图 3.8。这个最佳比对包括两对空格。在这个例子里，空格对的数量等于比对离开主对角线的距离，但在其他情况下并不一定这样。可以肯定的一点是：空格对的数量大于或等于离开对角线的最大偏离距离。

```
-GCGC-ATGGATTGAGCGA
TGCGCCATGGAT-GAGC-A
```

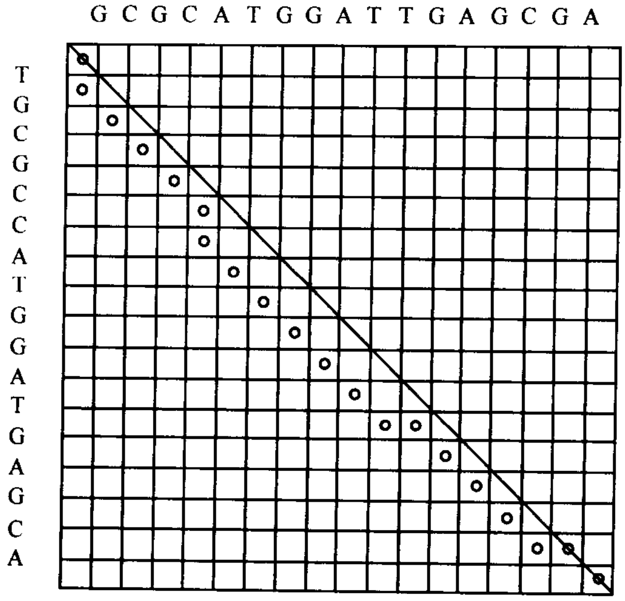


图 3.8　一个优化比对以及它在动态程序设计矩阵中的对应路径

基本概念是这样的：如果序列是相似的，最佳比对的路径就接近主对角线。为了计算优化比对与计分，我们不必填充整个矩阵，填充一个沿主对角线的狭窄带就足够了。

图 3.9 中算法 *KBand* 执行这个矩阵填充过程，沿主对角线填充一个宽为 $2k+1$ 的水平(或垂直)带。在结尾处，入口项 $a[n,n]$ 包含了比对的最高计分。这个算法运行时间为 $O(kn)$。如果 k 小于 n，这指标大大优于通常的 $O(n^2)$。

注意，我们没有使用任何 k 带(KBand)外的入口项，既不初始化它们，也不在最大值
67 计算中使用它们。为检查某个位置 (i,j) 是否在 k 带内，我们使用下面的标准：

$$InsideStrip(i,j,k) = (-k \leqslant i-j \leqslant k)$$

如同在其他动态程序设计算法中一样，每个入口项 $a[i,j]$ 要依赖于 $a[i-1,j]$、$a[i-1,j-1]$ 和 $a[i,j-1]$。在这里的代码里，我们不必检查位置 $(i-1,j-1)$，因为它与 (i,j) 在同样的对角线上，因此它将总是在带状区内。对 $(i-1,j)$ 和 $(i,j-1)$ 则需检查，因为当 (i,j) 在边界上时它们可能在带状区外。

68 如何使用算法 *KBand*？我们要选择一个 k 值运行这个算法。如果 $a[n,n]$ 大于或等于某个含不少于 $k+1$ 个空格对的比对的最佳计分，则我们幸运地用仅仅 $O(kn)$ 的时间就发现了一个优化比对。假如至少有 $k+1$ 个空格对，这个可能的最佳计分便是

```
Algorithm KBand
    input: sequences s and t of equal length n, integer k
    output: best score of alignment at most k diagonals away from main diagonal
    n ← |s|
    for i ← 0 to k do
        a[i, 0] ← i × g
    for j ← 0 to k do
        a[0, j] ← j × g
    for i ← 1 to n do
        for d ← −k to k do
            j ← i + d
            if 1 ≤ j ≤ n then
                // compute maximum among predecessors
                a[i, j] ← a[i − 1, j − 1] + p(i, j)
                if InsideStrip(i − 1, j, k) then
                    a[i, j] ← max(a[i, j], a[i − 1, j] + g)
                if InsideStrip(i, j − 1, k) then
                    a[i, j] ← max(a[i, j], a[i, j − 1] + g)
    return a[n, n]
```

图 3.9 主对角线周围 k-带状区的算法

$$M(n-k-1)+2(k+1)g \tag{3.9}$$

这个值是按恰有 $k+1$ 个空格对且其他字符对均匹配的假设计算的。这里 M 是一个匹配的计分，g 是每个空格对的值。我们假定 $M>0$ 和 $g\leqslant 0$。

如果 $a[n,n]$ 小于式(3.9)的值，我们将 k 乘 2 并再次运行。如果 k 的初始值是 1，后续值将是 2 的幂。终止条件是：

$$a_k[n,n] \geqslant M(n-k-1)+2(k+1)g$$

它等价于

$$k \geqslant \frac{Mn - a_k[n,n]}{M-2g} - 1$$

此时，我们已经运行了算法多次，每次用一个更大的 k。假定用 2 的幂，则总的复杂性是：

$$n + 2n + 4n + \cdots + kn \leqslant 2kn$$

为了控制整个复杂性，我们需要一个 k 的上界。如果我们不提早终止，则有

$$\frac{k}{2} < \frac{Mn - a_{k/2}[n,n]}{M-2g} - 1$$

如果 $a_k[n,n]=a_{k/2}[n,n]$，那么这就是优化记分 $\mathrm{sim}(s,t)$，而我们的边界是

$$k < 2\left(\frac{Mn - \mathrm{sim}(s,t)}{M-2g} - 1\right)$$ 69

如果 $a_k[n,n]>a_{k/2}[n,n]$，则优化比对有多于 $k/2$ 个空格对，因此

$$\mathrm{sim}(s,t) \leqslant M\left(n - \frac{k}{2} - 1\right) + 2\left(\frac{k}{2}+1\right)g$$

这导致

$$k \leqslant 2\left(\frac{Mn - \mathrm{sim}(s,t)}{M - 2g} - 1\right)$$

这和上面的边界几乎相同。

注意到 $M-2g$ 是个常数，时间复杂性因此是 $O(dn)$，d 是最大可能计分 Mn（两个相同序列的计分）和优化计分之间的差。因此，相似性越高，比对越快。

把这算法推广到不等长的一般序列对是直截了当的。空间节省版本亦可很容易地推导出来。

3.4 比较多个序列

到目前为止我们在本章中只讨论了两个序列的比较。然而，我们经常需要以最好的方式同时比较多个序列。例如，当我们面对许多不同生物的具有相似功能的蛋白质时，就遇到这个问题。我们可能想知道这些序列的那些部分是相似的，那些部分是不同的。为获得这些信息，我们需要对这些序列进行多序列比对(multiple alignment)，这就是本节的论题。

多序列比对是两序列比对的自然推广。令 $s_1,\dots,s_k$ 是同一字母表上的一组序列。一个包含 $s_1,\dots,s_k$ 的多序列比对可通过在序列中插入空格使得它们具有相同长度来进行。通常把扩展的序列放在一个垂直表中，使得对应位置上的字符以及空格位于同一列。我们进一步规定任何列不得仅由空格组成。图 3.10 显示了一个包含 4 个短氨基酸序列的多序列比对（多序列比对对蛋白质更常见，因此在本节的一些例子中我们使
70 用氨基酸序列）。

```
MQPILLL
MLR-LL-
MK-ILLL
MPPVLLL
```

图 3.10 4 个氨基酸序列的多序列比对

多序列比对中的一个重要问题是对比对品质的精确定义。我们下面将介绍一种基于序列对比对的多序列比对计分方法。此外，不同于把序列对叠加，科学家还探讨了把多个序列置入树状结构的多序列比对方法，这导致了对比对品质不同的度量。

3.4.1 SP 度量

多序列比对的计分比两序列比对要复杂。我们这里只限于讨论纯累加性函数，即比对计分是各列计分的和。记分方法要赋予每列一个计分值，然后累加这些记分值得到比对计分。然而，要对一个列计分，我们需要一个带 k 个自变量(argument)的函数，k 是序列的个数，每个自变量为字符或为空格。一种方法是建立一个 k 维阵列，它可用自变量寻址并返回值。这个方法的问题是需要为自变量的每种组合赋予一个值，而组合的数量可达 2^k-1 之多。一个典型的 k 值是 10，而这将导致超过1000种的组合。因此需要其他更便利的方法来定义这个函数。

便利的方法可通过确定这样的函数所应具有的“合理”特性而得到。首先，函数必须独立于自变量的阶。例如，如果一个列有 I，－，I，V，另一个列有 V，I，I，－，它们应当具有

相同的计分。其次，应该有一个函数酬奖相同或高度相关的残基并减罚无关残基及空格。配对和函数(sum-of-pairs function，SP 函数)是一个满足这些特性的函数，它定义为列中所有符号对的配对计分的和。例如，上述内容列的计分是：

$$SP\text{-}score(\mathrm{I},-,\mathrm{I},\mathrm{V}) = P(\mathrm{I},-)+P(\mathrm{I},\mathrm{I})+P(\mathrm{I},\mathrm{V})+P(-,\mathrm{I})+P(-,\mathrm{V})+P(\mathrm{I},\mathrm{V})$$

这里 $p(a,b)$是符号 a，b 的配对计分。当 a 或 b 是一个空格时，它包含一个空格减罚。这是一个十分方便的方式，因为它依赖于配对计分，就如同两序列比对中的配对计分。由于简单有效，SP 计分系统被广为使用。

一个需加以特别说明的小而重要的细节是，尽管不能有仅由空格构成的列，但一列中可以有两或多个空格。当计算 SP 计分时，我们需要一个对应于 $p(-,-)$的值，它在两序列比较中是没有的，因为不会出现这种情况，而基于 SP 的多序列比对就需要。通常的做法是让 $p(-,-)=0$，这看起来有点奇怪，因为空格通常是要被减罚的(即，每当有空格，配对计分是负数)，而配对的空格更应减罚。然而，存在足够的理由把 $p(-,-)$定义为 0，其中之一与配对比对本身有关。对一个多序列比对，我们经常根据其包含的配对比对得出结果。实际上，对任何多序列比对，我们可以选择两个序列而忽略其他序列，考察这两个序列的比对方式。不难发现，把由空格构成的列除外，这两序列间的比对实际是一个配对比对。而我们紧接着就可以移去这些空格，导出一个真正的配对比对，图 3.11 给出了一个例子。这就是我们所说的导出的配对比对(induced pairwise alignment)，或多序列比对的投影(projection of a multiple alignment)。 71

多序列比对

```
1 PEAALYG RFT---IKSDVW
2 PEAALYG RFT---IKSDVW
3 PESLAYNKF---SIKSDVW
4 PEALNYG RY---SSESDVW
5 PEALNYGWY---SSESDVW
6 PEVIRMQDDNPFSFQSL VY
```

取序列 2 和 4

```
PEAALYGRFT---IKSDVW
PEALNYGRY---SSESDVW
```

移去带空格的列

```
PEAALYGRFT-IKSDVW
PEALNYGRY-SSESDVW
```

图 3.11　导出的配对比对(投影)

仅当我们置 $p(-,-)=0$ 时，一个非常有用的事实才成立，即以下的关于多序列比对 α 的 SP 计分的公式：

$$SP\text{-}score(\alpha) = \sum_{i<j} score(\alpha_{ij}) \tag{3.10}$$

这里 α_{ij}是由 α 对序列 s_i 与 s_j 导出的配对比对。这个算法是正确的，因为它反映了做同一件事的两个途径。我们可以计算每一列的计分，然后相加所有列的计分；也可以计算每个导出的配对比对，然后累加它们的计分。在任一情形，对每一列 c 和每一字符对(i,j)，我们都累加记分 $p(s_i'[c],s_j'[c])$，这里 s'指扩展序列(插入了空格)。注意仅当 $p(-,-)=0$ 时这才成立。

==========动态程序设计==========

决定了一种度量或记分来确定多序列比对的品质后，我们现在来计算有最大计分的比对，即优化比对。

这里仍可使用两序列比对中的动态程序设计。简单起见，我们有 k 个长均为 n 的序 72

列。我们用一个长为 $n+1$ 的 k 维阵列 a 来存储序列前缀比对的优化计分。因此，$a[i_1,\ldots,i_k]$存储 $s_1[1\ldots i_1],\ldots,s_k[1\ldots i_k]$的优化比对的计分。

用0初始化 $a[0,\cdots,0]$之后，我们必须填写整个阵列。存储这个阵列的空间开销是 $O(n^k)$，这也是计算时间的下限，因为我们必须计算每一个入口项的值。由于如下理由，实际时间复杂性要更高。首先，每个入口项依赖于先前计算的 2^k-1 个入口项，当前列符号的每一种组合对应其中一项。在组合过程中，每个序列的参与者是一个字符或是一个空格。因为有 k 个序列，存在 2^k 种符号组合。除去被禁止的由全空格组成的列，最后得到 2^k-1 个。因此已有的指数时间复杂性 $O(n^k)$中又加入了一个因子 2^k。

接下来的问题是访问阵列中的数据。极少有程序设计语言能让用户在运行时定义任意 k 维的阵列。一个选择是实现自己的访问方式，不管有没有语言支持，我们希望能用 $O(k)$步完成每次访问。

另一个问题是对列计分的计算。SP 方法需要 $O(k^2)$步计算每一列，因为总计有 $k(k-1)/2$ 个配对比对计分（例如，3 列有 3 种，aa，ab，bc；4 列有 6 种，ab，ac，ad，bc，bd，cd）。更简单的办法，例如仅仅计数非空格符号数，需要至少 $O(k)$步，因为我们必须逐个检查所有自变量。

最后，是关于 $a[i_1,\ldots,i_k]$的实际计算，这涉及一个最大值计算。使用粗字体表示 k-元组，我们必须执行的运算可写为

$$a(\boldsymbol{i})\leftarrow\max_{b\neq 0}\{a[\boldsymbol{i}-\boldsymbol{b}]+SP\text{-}score(Column(\boldsymbol{s},\boldsymbol{i},\boldsymbol{b}))\}$$

这里 b 的取值范围是 k 个成分的所有非 0 二进制值，且

$$Column(\boldsymbol{s},\boldsymbol{i},\boldsymbol{b}))=(c_j)_{1\leqslant j\leqslant k}$$

$$c_j=\begin{cases}s_j[i_j] & if\ b_j=1\\ - & if b_j=0\end{cases}$$

如果使用 SP，全部运行时间是 $O(k^2 2^k n^k)$；如果列计分能在 $O(k)$时间内算出，则是 $O(k2^k n^k)$。如果 k 是固定的值，k 重嵌套的 for 语句能用于填充阵列。通过一个类似于配对比对中用到的回溯过程（见 3.2.1），我们能够在阵列中找到优化比对。对于序列不等长的例子，直截了当对序列的扩展能产生一个类似的算法。在任何情况下，算法的时间复杂性与输入序列的个数呈指数相关，多项式时间算法不存在，因为我们已经阐明使用
73 SP 方法的多序列比对是 NP-完全问题（见文献提要）。

==========节 省 时 间==========

纯动态程序设计算法的指数复杂性使得它不便于广泛应用。主要的问题是阵列的规模，对 3 个序列，我们就有 $O(n^3)$个单元。随着序列数量 k 的增加，我们有越来越大的单元数量需要填充。显然，如果我们能够在某种程度上减少所需计算的单元量，这将对处理时间有显著的影响。

这一部分描述一个启发式，它正是起这种作用。我们将说明如何将它置入动态程序设计算法以加速计算。之所以说它是一个启发式，是因为最坏情况下所有单元均需计算。但实际上良好的加速效果是可得到的。该启发式建立在多序列比对与两序列投影的关系

上，特别是它使用公式(3.10)联系 SP 计分与配对比对计分。因此，该方法仅适用于 SP 计分情形。

方法的轮廓是这样的。我们有 k 个长为 n_i 的序列，$1\leqslant i\leqslant k$，需要根据 SP 度量计算优化比对。动态程序设计仍被使用，但现在不处理所有的阵列单元，而仅仅处理与优化比对“相关”的单元。究竟哪些单元是相关的？为何相关？

答案来自对单元配对投影的考察。在预处理阶段，我们进行任意单元相关性的测试。为了利用这个测试并减少需要查看单元的数量，我们还需修改单元的填充次序。

= = = = = = = = = = = 相关性测试 = = = = = = = = = = =

令 α 是包含串 $s_1,\ldots,s_k$ 的一个优化比对。我们首先须明白，α 虽是优化比对，但对于给定序列对，它的投影并不一定是最佳的。图 3.12 是一个其投影比对非最佳比对的例子。而不幸的是这种情况确实会发生，否则我们本可以容易地进行相关单元的检测，即若一个单元的每个配对投影都是对应于该投影的两个序列的优化比对的一部分，则该单元是相关单元。

优化多序列比对	非优化投影
AT	
A-	A-
-T	-T
AT	
AT	

图 3.12　带非优化投影的优化比对

在继续讨论之前，我们说明，可以方便地使用我们前面介绍过的双序列比较算法产生一个矩阵， 74
其每个入口项 entry(i,j)存记了含 $cut(i,j)$的比对的最高计分值。当配对比对 α 能被分成两个子比对时——一个是 $s[1\ldots i]$与 $t[1\ldots j]$之间的比对，另一个是 s 的剩余部分与 t 的剩余部分的比对，α 就包含了一个 $cut(i,j)$。

为获得所需值，我们加入两个包含如下内容的动态程序设计矩阵 a 和 b：

$$a[i\ldots j]=\mathrm{sim}(s[1\ldots i],t[1\ldots j])$$

$$b[i,j]=\mathrm{sim}(s[i+1\ldots n],t[j+1\ldots m])$$

这里 $n=|s|,m=|t|$。矩阵 a 是我们一直使用的标准矩阵，矩阵 b 的计算如同矩阵 a，但方向相反，先初始化最后一行和一列，然后反向计算直至到达 $b[0,0]$，我们在 3.3.1 讨论动态程序设计的线性空间实现算法时已进行了描述。函数 *BestScoreRev* 完成我们在 b 中所要进行的计算。然后，矩阵 $c=a+b$ 恰好包含了在(i,j)切下的比对的最高计分。我们把 c 称做总分矩阵，而 a 和 b 分别称做前缀计分矩阵和后缀计分矩阵。

矩阵 c 比 a 或 b 更引人注目。用矩阵 c 我们能快速找出最佳比对，而用矩阵 a 与 b 完成这件事则相当困难。我们用一个例子进行解释。在图 3.13 中，对一个确定的序列对以及确定的计分系统我们有矩阵 a 和 c。靠查询矩阵 a 你能准确看出最佳比对所在吗？用矩阵 c 则容易的多，因为我们仅需查看具有最高计分值的单元。

尽管优化比对的投影本身可能不是优化的，但只要我们有优化计分的下界，我们就可以为投影计分规定一个下界。下面的结果可以说明其机制。

定理 3.1　令 α 是一个包含序列 $s_1,\ldots,s_k$ 的优化比对。如果 $SP\text{-}score(\alpha)\geqslant L$，则

$$score(\alpha_{ij})\geqslant L_{ij}$$ 75

这里

		G	A	T	T	C
	0	−2	−4	−6	−8	−10
A	−2	−1	−1	−3	−5	−7
T	−4	−3	−2	0	−2	−4
T	−6	−5	−4	−1	1	−1
C	−8	−7	−6	−3	−1	2
G	−10	−7	−8	−5	−3	0
G	−12	−9	−8	−7	−5	−2

		G	A	T	T	C
	−2	−2	−7	−12	−17	−22
A	−7	−4	−2	−7	−12	−17
T	−10	−7	−5	−2	−7	−12
T	−13	−10	−7	−5	−2	−7
C	−14	−13	−10	−5	−4	−2
G	−17	−14	−13	−8	−4	−2
G	−22	−17	−14	−11	−7	−2

图 3.13　动态程序设计矩阵:前缀计分(左)与总计分(右)

$$L_{ij} = L - \sum_{\substack{x<y \\ (x,y)\neq(i,j)}} (\mathrm{sim}(s_x, s_y))$$

证明:我们相继有

$$SP\text{-}score(\alpha) \geqslant L,$$

$$\sum_{x<y} score(\alpha_{xy}) \geqslant L,$$

$$\sum_{\substack{x<y \\ (x,y)\neq(i,j)}} (score(\alpha_{xy})) \geqslant L - score(\alpha_{ij}),$$

$$\sum_{\substack{x<y \\ (x,y)\neq(i,j)}} (\mathrm{sim}(s_x, s_y)) \geqslant L - score(\alpha_{ij})$$

结论由此得证。

我们现在可以开始测试,相对于下界 L,一个下标为 $i=(i_1,\dots,i_k)$的单元是否与优化比对相关。简单地说,如果它的所有投影满足上述定理的条件,则这个单元是相关的。换言之,i 是相关的,如果

$$c_{xy}[i_x, i_y] \geqslant L_{xy},$$

对所有 x 与 y 都有 $1 \leqslant x < y \leqslant k$,这里 c_{xy}是 s_x 和 s_y 的总计分矩阵。根据定理,只有相关单元参与优化比对,尽管不是所有相关单元均参与优化比对。这减少了参与优化比对的潜在单元的数量。当 L 接近真正的优化计分值时,这一减少更为突出。

为获得合适的边界 L,我们可以选择包含所有序列的任意一个多序列比对并把它的计分作为 L。当然这是一个下界,因为优化比对才有最高的计分。如果我们已经有了一个合理的好的比对,我们可以利用它的计分并改进比对,从而得到一个优化比对。另一个可选用的下界可利用 3.4.2 段的结果获得。

＝＝＝＝＝＝＝＝＝＝＝实 现 细 节＝＝＝＝＝＝＝＝＝＝＝

在实现上述启发式算法时我们需要谨慎。测试所有单元的相关性然后仅仅处理相关单元是不够的,由于这样仍须处理(测试)每个单元,不能导致显著的时间节省。我们需要
76 一个完全削除无关单元的办法,以便不需再处理它们。

这里是一个可能的策略。我们从位于 $\mathbf{0}=(0,0,\cdots,0)$的单元开始,它总是相关的,并影响依赖于它的单元。每个这样的单元又有它自己的受影响单元,如此推展,直至达到在最终角落的单元$(n_1,\dots,n_k)$。在整个过程中,只有相关单元才被处理。

为了更好地解释这个策略,我们需要几个定义。如果单元 i 是被用于计算 $a[j]$值的

单元之一，就说单元 i 影响单元 j。这里，我们也说单元 j 依赖于单元 i。这一情形的另一个特征是 $b = j - i$ 是一个向量，其每个元素为 0 或 1，且 $i \neq j$。因此，每个单元依赖于至多 $2^k - 1$ 个其他单元，亦至多影响 $2^k - 1$ 个其他单元。我们说"至多"是因为边界单元可能影响或依赖少于 $2^k - 1$ 个其他单元。

我们保持一个待测单元池，开始时，仅单元 **0** 在池中。在任何时候单元池仅包含相关单元。当单元 i 进入单元池时，其值 $a[i]$ 即被初始化。在停留在池的过程中，其值被更新。当这个单元被移出池时，$a[i]$ 的当前值成为单元 i 的真正值，并被用来找出依赖于它的相关单元。

一个单元的值是按如下方式传播的。令 j 是一个依赖于 i 的相关单元。如果 j 不在池中，则将其放入，并用下式初始化其值

$$a[\boldsymbol{j}] \leftarrow a[\boldsymbol{i}] + SP\text{-}score(Column(\boldsymbol{s}, \boldsymbol{i}, \boldsymbol{b}))$$

如果 j 已在池中，则用下式更新其值

$$a[\boldsymbol{j}] \leftarrow max(a[\boldsymbol{j}], a[\boldsymbol{i}] + SP\text{-}score(Column(\boldsymbol{s}, \boldsymbol{i}, \boldsymbol{b})))$$

重要的一点是在每个时刻总是某个词序最小的单元被选中移出池，这保证了该单元已接收了来自相关单元的影响，其值不会再被更新。这个过程在到达单元 $(n_1, \ldots, n_k)$ 时停止，它是最后一个需被检测的单元，其值就是所求的 SP 计分。计算计分的算法见图 3.14。为找到优化比对，我们需以某种方式记下更新过程。一个办法是构造一个依赖图，图中相关单元是节点，边则代表提供了最大值的影响。这个算法的时间与空间复杂性与相关单元的数量成比例。 77

```
Algorithm Multiple-Sequence Alignment
  input: s = (s_1, ..., s_k) and lower bound L
  output: The value of an optimal alignment
  // Compute L_xy, 1 ≤ x < y ≤ k
  for all x and y, 1 ≤ x < y ≤ k do
      Compute c_xy, the total score array for s_x and s_y
  for all x and y, 1 ≤ x < y ≤ k do
      L_xy ← L − Σ_{(p,q)≠(x,y)} sim(s_p, s_q)
  // Compute array a
  pool ← {0}
  while pool not empty do
      i ← the lexicographically smallest cell in the pool
      pool ← pool \ {i}
      if c_xy[i_x, i_y] ≥ L_xy, ∀x, y, 1 ≤ x < y ≤ k, then // Relevance test
          for all j dependent on i do
              if j ∉ pool then
                  pool ← pool ∪ {j}
                  a[j] ← a[i] + SP-score(Column(s, i, j − i))
              else
                  a[j] ← max(a[j], a[i] + SP-score(Column(s, i, j − i)))
return a[n_1, ..., n_k]
```

图 3.14　带启发式省时措施的多序列比较动态程序设计算法

3.4.2 星比对

即便应用前面描述的省时措施，使用标准动态程序设计方法计算多序列优化比对也费时冗长。因此人们又提出了其他方法，这些候选方法多是启发式的，即它们不保证比对结果的质量，而仅是更快地获得答案。在许多情况下，所得的答案是合理的好答案。

其中一个方法是星比对，它是基于一个固定序列与所有其他序列的配对比对而建立的。这个固定序列是星的中心，比对 α 的构造要求是：当 i 或 j 是中心序列的下标时 α 的投影 α_{ij} 是优化的。

令 $s_1, \ldots, s_k$ 是需比对的 k 个序列。为构造一个星比对，我们首先需挑选一个中心序列（译者注：存在另一个构造星比对的方法，允许中心序列不在输入序列集合中。例如，中心序列可以是与所有其他输入序列具有最大配对相似性计分的序列。在这个意义上，星比对可看做是树比对的一个特例）。这里先暂不讨论中心序列如何挑选。假定中心序列已被选出，其下标是 1 到 k 之间的数 c。接着，我们对每个下标不等于 c 的序列作 s_c 与 s_i 的优化比对，这可由标准动态程序设计实现，费时 $O(kn^2)$，假定所有序列长为 $O(n)$。

然后，我们使用一种称为"一旦为空格，始终为空格"的技术将这些配对比对向 s_c 汇集。例如，构造过程可从 s_1 与 s_c 的配对比对开始，接着不断加入配对比对。在这个过程中，我们不断地往 s_c 中加入空格以适配比对，且决不移出已加入的空格，即空格一旦被加入 s_c 中，就始终留在 s_c 中。

随着后续配对比对的不断加入，一方面我们有一个包含 s_c 序列以及若干其他序列的多序列比对，另一方面我们有 s_c 与一个新序列之间的配对比对。在两种比对中我们插入尽可能少而必要的空格，以保持与扩展的 s_c 序列相适配。然后，将新扩展的序列加入序列群中，且它和其他已扩展的序列具有相同长度。

78 这种加入操作的时间复杂性取决于数据结构，在合理的数据结构下它不高于 $O(lk)$，l 是比对长度的上界。因为我们有 $O(k)$ 个序列需加入，加入结束时费时 $O(k^2 l)$。整个时间复杂性是 $O(kn^2 + k^2 l)$。如果我们要知道最终计分，则需额外花 $O(k^2 l)$ 的时间，但渐进时间复杂性仍旧相同。

如何选择中心序列？一种方法是逐个测试，择优而取；另一种方法是计算全部 $O(k^2)$ 个优化配对比对，选择使

$$\sum_{i \neq c} \text{sim}(s_i, s_c) \tag{3.11}$$

最大的序列为中心序列。

例 3.1　考虑如下5个 DNA 序列。我们由构造一个显示序列间配对相似性的表开始（图 3.15），计分系统与 3.2.1 相同。

s_1 = A T T G C C A T T
s_2 = A T G G C C A T T
s_3 = A T C C A A T T T T
s_4 = A T C T T C T T
s_5 = A C T G A C C

	s_1	s_2	s_3	s_4	s_5
s_1		7	−2	0	−3
s_2	7		−2	0	−4
s_3	−2	−2		0	−7
s_4	0	0	0		−3
s_5	−3	−4	−7	−3	

图 3.15　例 3.1 中序列的配对计分

从表中可看到,第一个序列 s_1 是使表达式(3.11)最大化的序列。下一步是确定 s_1 与所有其他序列之间的优化比对。假定我们选择了如下的优化比对:

```
ATTGCCATT
ATGGCCATT

ATTGCCATT--
ATC-CAATTTT    79

ATTGCCATT
ATCTTC-TT

ATTGCCATT
ACTGACC
```

在这个例子中,中心序列 s_1 仅在与 s_3 的比对中引进了 s_1 结尾处的两个空格,因此在最后的多序列比对中 s_1 至多只有这两个空格。其他的序列按所选的比对与 s_1 对齐,最后的结果是:

```
ATTGCCATT--
ATGGCCATT--
ATC-CAATTTT
ATCTTC-TT--
ACTGACC----
```

3.4.3　树比对

本段介绍多序列比对的另一种方法。提出这个方法的原因是,有时需比对的序列可构成一棵进化树(进化树见第 6 章),在这种情况下我们能够根据树的边所对应的配对比对计算整体相似性。而与之相比,SP 度量则考虑所有配对比对的相似性。

假定有 k 个序列和一个恰有 k 个叶子的树,树叶与序列间有一一对应性。如果我们把序列指派给树的内部结点,我们就能计算每个边的权,它是与该边相连的结点所对应的两序列间的相似性。所有边的权的和是树的计分。找出一个能使该计分最大的序列指派就是所谓的**树比对**(tree alignment)。星比对可看做是树比对的一个特例,其中的树是一个星。

例 3.2　考虑一个如图 3.16 的输入,指派序列 CT 给顶点 x,序列 CG 对顶点 y,我们得到一个计分 6。这个计分系统是:如果 $a=b$ 则 $p(a,b)=1$,否则为 0;$p(a,-)=-1$。

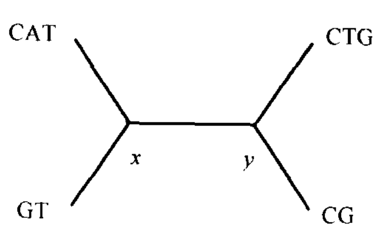


图 3.16　一个树比对问题的输入

树比对问题是 NP-难题。存在一个发现优化解的算法,但它与序列的个数呈指数关系。使用 3.4.1 中发展的方法,我们有可能改进时间与空间需求。当边的权用距离而非相似性定义时,也存在能够保证良好性能的近似算法(见 3.6.1)。在本例中我们寻找一个序列指派使得距离和最小,有关结果可见文献提要。 80

3.5　数据库搜索

随着快速可靠的核酸与蛋白质测序技术的出现，人们建立了集中化的数据库来存储世界各地实验室产生的大量测序数据，这要求有高效的程序对这些数据库进行查询。在一个典型的查询应用里，我们有一个*查询序列*(query sequence)，用以与数据库中所有序列进行比较以搜寻局部相似性，而这便意味着对几十万个序列进行比较。

我们迄今为止所讨论的相似性计算和两序列优化比对的方法都具有平方时间复杂性，这使得它们无法适用于大规模数据库的搜索。为加速搜索，人们提出了新的更快的方法。一般地，这些方法是基于启发式的，因此难以建立理论上的时间与空间复杂性。不过基于这些方法的程序已成为非常重要的工具，因此值得仔细研究。

在这一部分我们集中讨论两个最流行的数据库搜索程序。尽管其中之一使用了动态程序设计方法的一个变体，但两者均未用纯粹的动态程序设计。

在开始描述这些程序之前，我们先解释某些氨基酸计分矩阵的基础知识，它们在数据库搜索和蛋白质序列比较中非常重要。

3.5.1　PAM 矩阵

当比较蛋白质序列时，简单计分系统(如 +1 表示匹配，0 表示失配，-1 表示空格)是不够的。构成蛋白质的氨基酸残基具有不同的生物化学特性，这些特性可影响它们在进化过程中的相互替换性。例如，与体积差异大的氨基酸相比，体积相似的氨基酸更易于彼
81 此替换。其他特性，诸如与水的亲和性，亦影响相互替换的概率。因为作蛋白质比较时常常需考虑进化因素，使用一个能够充分反映这些特性的计分系统就十分重要。

影响相互替换概率的因素很多，因此对实际替换率的直接观察常常是导出残基对之间相似性计分的最好方法。针对这一目的的一个标准过程基于一个重要的计分矩阵家族，即 PAM 矩阵，PAM 指的是 Point Accepted Mutation(可接受点突变)或 Percent of Accepted Mutation(可接受突变百分比)。基础的 1-PAM 矩阵反映的是进化产生的每百个氨基酸平均发生一个突变的量值。在这一部分我们简要介绍如何计算 PAM 以及它的意义。

我们的描述略异于该矩阵原初的思想，但突出了氨基酸频率和氨基酸突变性的作用。特别是，突变性的定义在不同的情况下可以是不同的。

首先，用户必须选择一个用于序列比较的进化距离，记分矩阵是这个距离的函数。例如，一个 250-PAM 矩阵适合于对两个进化距离为 250 单位的序列进行比较。在本书的推导里我们首先构造对应于 1-PAM 的矩阵，然后由它获得其他进化距离的矩阵。此外，仅考虑在氨基酸层次的突变，不考虑 DNA 层次的突变。

对每一个进化距离我们有一个*概率转移矩阵*(probability transition matrix) M 和一个*计分矩阵*(score matrix) S。计分矩阵是由概率矩阵得到的，因此我们从概率矩阵谈起。建立 1-PAM 矩阵 M 的必备要素是：

- 可接受突变表

- 每个氨基酸 a 的出现概率 p_a

一个可接受突变是一个发生了且被环境正性(positive)选择的突变,即它没有导致发生突变个体的死亡。收集可接受突变的一个办法是比对两个来自不同种属的同源蛋白质,例如对人和猩猩的血红蛋白 α 链,每个字符不同的位置都是一个可接受突变。我们把这些可接受突变看做无方向性事件,即给定一对氨基酸 a 和 b,我们不能肯定这二者中是谁发生了突变。不论在始祖序列里该位置是什么,我们都无法知道始祖序列。在始祖序列里该位置甚至可能是第三个氨基酸,它突变成现在的两个不同的氨基酸。对非常接近的序列而言这个可能性是不大的。注意,对基本的 1-PAM 矩阵我们只考虑直接的突变,即 $a \to b$,而不考虑有中介的突变,即 $a \to c \to b$。

通过在一个大且全面的蛋白质序列集上计算各种氨基酸出现的相对频率,我们可简单估算氨基酸的出现率,它满足:

$$\sum_a p_a = 1$$ 82

从可接受突变表我们可以计算 $a \leftrightarrow b$ 突变的已发生次数 f_{ab}。注意这里处理的是无方向突变,因此 $f_{ab} = f_{ba}$。求和式

$$f_a = \sum_{b \neq a} f_{ab}$$

是与 a 有关的总突变次数,而

$$f = \sum_a f_a$$

是突变涉及的氨基酸总数。显然 f 是突变总数的两倍(因每次突变涉及两个氨基酸)。

构造 1-PAM 概率转移矩阵 M 所需要的便是频率 f_{ab} 和概率 p_a。M 是一个 20×20 矩阵,矩阵单元 M_{ab} 是氨基酸 a 转变成氨基酸 b 的概率。注意 a 和 b 可以是相同的,它指的是在这段特定的进化间期 a 保持不变的概率。M_{aa} 的计算基于氨基酸 a 的相对突变性(relative mutability) m_a,即

$$m_a = \frac{f_a}{100 f p_a} \tag{3.12}$$

一个氨基酸的突变性是对其变化的度量,它是给定氨基酸在特定进化期间发生变化的概率。因此 a 保持不变的概率是突变概率的补

$$M_{aa} = 1 - m_a$$

另一方面,a 变为 b 的概率可按条件概率的积计算,即 a 突变的概率乘以 a 突变条件下 a 变成 b 的概率。我们把条件概率估算为 $a \leftrightarrow b$ 突变次数与 a 突变总数之间的比率,因此

$$\begin{aligned} M_{ab} &= \Pr(a \to b) \\ &= \Pr(a \to b \mid a \text{ 改变}) \Pr(a \text{ 改变}) \\ &= \frac{f_{ab}}{f_a} m_a \end{aligned}$$

注意,我们是用简化了的蛋白质进化模型来计算这些概率的。例如,我们假定氨基酸的突变与历史无关,而根据遗传编码的性质其实并非这样;我们也忽略了序列中其他氨基酸对突变氨基酸的影响。历史无关性导致进化的马尔可夫型模型,它具有良好的数学性

质，我们将在后续部分提到。

容易验证，M 具有如下性质：

$$\sum_b M_{ab} = 1 \tag{3.13}$$

83 $$\sum_a p_a M_{aa} = 0.99 \tag{3.14}$$

等式(3.13)指的是，对一个氨基酸 a，将 a 不变的概率加上 a 变化到每一个其他氨基酸的概率得到的值是 1。因此，我们有理由把这些数字(M_{ab}等)称做概率。这些概率指的是进化中氨基酸变化的单位。不应把进化单位看做时间单位，因为不同的事物按不同的速率变化，因此时间和进化的量值一般并不直接成比例。

在这个模型中，我们用平均每百个氨基酸发生一个突变的进化量作为进化单位，它称为 1-PAM 进化距离。概率转移矩阵被正则化用以反映等式(3.14)中的事实。我们在等式(3.12)的分母中乘以因子 100 来达到正则化。如果我们乘以 50 而非 100，则得到相同性质的矩阵，只不过式(3.14)反映的是每 50 个氨基酸发生一个突变的情况，即不变的概率是 0.98，这时一个进化单位将导致平均每 50 个氨基酸发生 1 个突变。

一旦有了基本矩阵 M 我们就能对更大的进化量导出转移概率。例如在两个 PAM 进化单位的条件下，a 转变为 b 的概率是多少？在第一个进化单位里 a 可以转变成任何氨基酸 c，包括它自身，概率是 M_{ac}，然后在第二个进化单位里 c 再以概率 M_{cb} 转变为 b。累加这些数字，我们知道最后的数字不大于 M_{ab}^2，这是 M 平方项的一个入口。一般来说，M^k 是进化单位 k 的概率转移矩阵。

一个有趣的事实是(可被证明)，当 k 变的非常大时(例如几千)，M^k 收敛于一个所有行均相等的矩阵，每一行包含列 b 中的相对频率 p_b。这就是说不论开始时是什么氨基酸，当这个长的进化过程结束时最终的氨基酸以概率 p_b 变为 b。

我们现在准备定义计分矩阵，这些矩阵的入口与两个概率的比率有关，一个是氨基酸的突变概率，另一个是氨基酸的随机出现概率，这个比率称做似然率或几率比率。

现在我们对一对氨基酸 a 和 b 计算这个比率。假定我们有比对 a 和 b，从 a 考虑，b 由于突变而在另一个序列中出现的概率是 M_{ab}，另一方面，b 还有一个随机出现的几率，这个比率因此是

$$\frac{M_{ab}}{p_b}$$

即使 b 等于 a 推理亦相同。实际记分是这个比率的对数值的 10 倍。在实践中为加速计算，实际值通常被舍入为最接近的整数，对数值乘以 10 正是用以减小正确值与整数近似值之间的差异。

前面的讨论基于 1-PAM，但我们可用完全相同的机制处理任意的进化距离。k-PAM 距离的计分矩阵定义如下

84 $$score_k(a, b) = 10\log_{10}\frac{M_{ab}^k}{p_b}$$

初看之下似不明朗，但这实际上是一个对称矩阵。计分矩阵必须是对称的，以便产生一个不依赖序列次序的比较计分。对 $k=1$，对称性不难验证，因为

$$\frac{M_{ab}}{p_b}=\frac{f_{ab}m_a}{f_a p_b}=\frac{f_{ab}}{100fp_a p_b}$$

对于更大的 k 值，可以证明对称性仍成立（习题 15），它源于这样的事实，即可接受突变是无向的，因此 $f_{ab}=f_{ba}$ 既用于计算 a 变到 b 的概率也用于计算 b 变到 a 的概率。

尽管这些记分矩阵只是根据可观察到的可接受突变产生的，它们也能够反映氨基酸的某些重要的化学和物理性质，诸如它们与水分子的亲和性和它们的大小。事实上，通常是具有相似性质的氨基酸具有高的配对记分，而无关氨基酸则有低的配对记分。这并不奇怪，因为相似氨基酸比无关氨基酸更易于产生可接受突变替换，且相似氨基酸之间的替换能够保持所涉及蛋白质的大多数性质。

我们在开头曾说需要固定 PAM 距离以得到一个计分矩阵进行比较。但有时我们有两个序列且对它们的进化距离一无所知，在这种情况下宜使用两个或三个覆盖较大范围的矩阵进行比较，如 40-PAM，120-PAM，250-PAM。总体上，低 PAM 适于发现短而强的局部相似性，高 PAM 适于找出长而弱的相似性。

3.5.2 BLAST

在这一部分我们介绍 BLAST 家族，它是最常用到的序列相似性比较的工具，用于序列数据库的搜索。BLAST 是 Basic Local Alignment Search Tool（基本局部比对搜索工具）的缩写。

数据库在这里仅仅是指大规模分类序列集，并不包含任何标准数据库管理系统中所拥有的数据访问、数据共享等操作。对我们而言，"数据"仅仅就是序列集合，尽管这些序列信息可能包含大量的附加信息如数据起源、文献索引、序列功能（如果已知）等等。

BLAST 返回一个查询序列与数据库序列之间的高计分片段对表。在解释该表的意义以及如何获得它之前我们先简要介绍片段对的某些术语。

一个片段是序列的一个子串。给定两个序列，它们之间的一个*片段对*（segment pair）是一对等长的分别来自两个序列的片段。因为片段对中的子串具有相同长度，我们能够 85
构造它们的一个无空隙比对，这个比对能够用一个氨基酸替换计分矩阵进行计分，无需空隙罚分函数（因为没有空隙）。通过片段对的计分可得到 BLAST 的计分。一个用 PAM120 替换矩阵计分的片段对见图 3.17，片段对是基本无空隙的局部比对。

K	A	L	M	R		
V	A	K	N	S		
−4	3	−4	−3	−1	→	总计：−9

图 3.17 片段对以及在 PAM120 下的计分

我们现在可以精确描述 BLAST 了。给定一个查询序列，BLAST 返回查询序列与数据库序列所有计分值大于某阈值 S 的片段对。参数 S 由系统提供，亦可由用户确定。我们已提到这种比对的一个特征是没有空隙，这实际上正是为什么 BLAST 运行如此之快的一个主要原因，因为对带空隙序列查找一个好的比对十分费时。

两个序列间的*最大片段对*（maximum segment pair，MSP）是计分值最大的片段对，这

个计分是序列相似性的一种度量，能够用动态程序设计算法精确计算。然而 BLAST 用一种比任何动态程序设计方法更快的办法估算这个值，这个方法也返回局部最大片段对，即无论扩展或缩短均不能使计分增大的片段对。

＝＝＝＝＝＝＝＝＝＝＝BLAST 工作原理＝＝＝＝＝＝＝＝＝＝＝

计算高计分片段对的 BLAST 方法如下。它找出某些"种子"，即查询序列与数据库序列间非常短的片段对，然后向两端不带空隙地扩展这些种子，直到达到最大可能计分。程序并不寻找所有的扩展，当计分低于某个既定的标准时便停止。由于这种时间上的优化，存在一个很小的机会使得正确的扩展未被找到。但实践证明这种取舍是完全值得的。

我们可把 BLAST 看做一个三阶段算法过程：

1. 编辑高计分串表(BLAST 的术语是字)；
2. 查找命中(hit)，每个命中产生一个种子；
3. 扩展种子。

86 特定的算法步骤取决于序列的种类是 DNA 还是蛋白质，我们将分别说明。

对于蛋白质序列，高计分字表由所有 w 个字符的字(称作 w-mer)组成，这些字与查询序列的某个 w-mer 的计分最少为 T，使用某个 PAM 矩阵计算计分。w 和 T 是程序参数。注意这个字表可能并未包括所有的查询序列 w-mer。如果一个查询序列 w-mer 由非常普通的氨基酸组成，它可能被剔除，因为即便它自身与自身的比对，其分值也可能低于 T。不过，算法亦可强制性地包括所有的查询序列 w-mer。对蛋白质搜索，推荐的 w 值(即种子的大小)是 4。

有两种方法搜寻数据库以搜索表中的命中。一种是把种子表中的字安排成哈希表(hash table)，然后对每个大小为 w 的数据库字，我们容易计算它在哈希表中的索引并与相应的字进行比较。另一种方法使用确定有限自动机查找命中，自动机有状态、转换、操作，它由确定的初始状态开始，数据库中的每一个字符导致一个转换，使当前状态变换到另一状态。根据状态以及转换，表中的一个字被识别出来。自动机仅构造一次，使用高计分字表作为输入，同时它也是一种存储这些字的紧凑方式。搜索是快速的，因为每个字符仅需一次转换。

最后的扩展直截了当。如前所述，为了节省时间，在某一点，当计分一定程度地低于由该点作最短扩展可获得的最佳计分时算法便停止，在两个方向均如此。由种子产生的高计分片段对被保存。对于重要扩展的漏失存在一个小的概率，但这是可忽略的。

对 DNA 序列，因为其计分更为简单，初始表仅仅包含查询序列 w-mer 便够了。搜寻策略也完全不同于蛋白质序列的情形。因为字母表的大小仅为 4，算法首先对数据库进行压缩，使得每个核苷酸表示为两个二进制数位，四个核苷酸构成一个字节。这样做除了节省了空间，搜索也得以更快，因为每次可比较一个字节(即四个核苷酸)。有一个额外的过滤步骤用于在数据库中滤去初始表中的极为普通的字，以避免大量的伪命中。

扩展以与蛋白质搜索类似的方式进行。在两种情形中扩展都以统计理论为依据，这个统计理论给出了随机序列无空隙局部最大计分的确切分布，并对由于偶然性发现片段对的概率进行精确的计算。该概率越小，匹配便越有意义。

我们简要勾画这个统计理论的要点。长度分别为 m 和 n 的随机序列 s 和 t 的 MSP 计分的分布可用下式近似计算，计算的精确性随着 m 和 n 的增大而提高。

对应于字母表中的字符对我们有一个替换开支矩阵 s_{ij}，对于每个字符在序列中的出现我们有一个出现概率 p_i。首先解下列方程计算一个值 λ

$$\sum_{i,j} p_i p_j e^{\lambda s_{ij}} = 1$$ 87

参数 λ 是该方程的惟一正解，能用牛顿方法解得。一旦 λ 已知，计分大于 S 的序列 s 与 t 之间的互不相同片段对的数量是

$$Kmne^{-\lambda s}$$

这里 K 是一个可计算常数。实际上，计分超过 S 的片段对的数量呈泊松分布，它的均值由上一个公式给出。从这一点我们易于导出其他有用值，如平均计分、计分落入 90% 时间的区间等等。

3.5.3 FAST

FAST 是另一个序列数据库搜索程序家族。第一个程序 FASTP 用于对给定查询序列按相似性搜索序列数据库。FASTP 的基本算法是依次比较数据库串与查询序列，报告具有显著相似性的串以及其他相关信息。因此，FASTP 的速度取决于其快速比较两个序列的速度。我们关注 FASTP 如何执行这一基本操作。

令 s 和 t 表示两个被比较的序列，且假定是蛋白质序列。它们的长度分别是 $m=|s|$ 和 $n=|t|$。比较由确定两者公共的 k 元组开始，$k=1$ 或 2。k 的值是一个参数，称做 ktup。此外，公共 k 元组的偏移量也是重要的，它是一个介于 $-n+1$ 到 $m-1$ 之间的值，决定两个序列的相对位移。特别是，如果公共 k 元组开始于位置 $s[i]$ 和 $t[j]$，我们说偏移量是 $i-j$。

算法需要下面的数据结构：①一个查询表(lookup table)；②一个以 0 为初始值，以偏移量为下标的向量。首先扫描序列 s，产生一个查询表，列出 s 中 k 元组的所有位置。然后扫描序列 t，同时在 s 的查询表中查找 t 的每个 k 元组。在偏移量向量的相应入口项中对公共出现的 k 元组进行累加。图 3.18 显示了偏移量向量的最后内容以及序列 $s=$ HARFYAAQIVL 和 $t=$ VDMA AQIA 的查询表。注意偏移量 +2 有最大的入口项值，意味着在这个偏移量上有许多匹配。这个方法称做对角线方法，因为偏移量可被看做是动态程序设计矩阵中的对角线。最大偏移量的一个可能用途是运行动态程序设计算法，该算法的对角线与该偏移量相对应。

FASTP 对公共 k 元组作更详细的分析，且当这些 k 元组在同一对角线上并相距不远时合并这些 k 元组。确切的标准是启发式的，这些合并的 k 元组构成所谓的区域(region)。区域可被看做 BLAST 中的片段对，或是无空隙局部比对。根据区域中所含的匹配或失配一个区域被赋予某个计分。重要的是记住区域无空隙。

下一步是对 5 个最佳区域用 PAM 矩阵进行重新计分，通常是 PAM120 或 PAM250。 88
这些新计分中的最佳者成为 s 和 t 之间相似性的一个初步度量，称做*初始计分*(initial score)。相对于一个查询序列，每个数据库序列都有一个初始计分，这些值用柱状图显示。

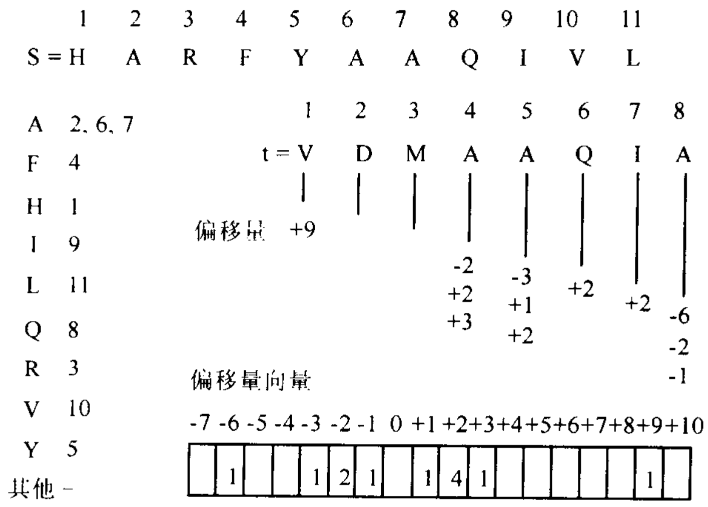


图 3.18 查询表和通过 FASTP 进行序列比较的偏移量向量。ktup 值为 1

初始计分还用于对所有数据库序列进行排序，对排序最高的若干序列，用动态程序设计算法产生一个优化计分。这个过程与 3.3.4 中介绍的十分类似。在实践中，当序列真正相关时，优化计分通常显著高于初始计分，这一事实有助于辨别偶然的好比对和真正的相关性。

ktup 的值在敏感性和选择性上影响算法的性能。敏感性指的是搜索工具识别出相关序列的能力，选择性指的是搜索工具排除假阳性的能力（假阳性是无关序列间的匹配）。一般地，敏感性和选择性是一对矛盾的目标。当使用 FASTP 或该家族中的其他程序时，小的 ktup 增加敏感性而大的 ktup 增加选择性。

除了搜索工具，FAST 家族还包括一个用来评估计分的统计意义的程序。这个工具打乱一个序列，例如 t，保持氨基酸组分不变但改变它们的次序，然后对重排后的 t 和 s 运行动态程序设计算法。重复此过程多次，以便得到平均计分和其标准偏差。根据这些值，按下式计算一个 z 值：

$$z = (\text{计分} - \text{平均计分}) / (\text{计分的标准偏差})$$

89 不过，由于随机序列相似性计分的统计分布不是正态分布，算出的 z 值用处有限。

FASTP 的改进导致 FASTA。一个增加的特征是可处理 DNA 和蛋白质两种序列，因此 FASTA 可视做 FASTP 和 FASTN 的组合，后者是专用于核酸序列的。该家族的另一个程序 TFASTA 用于蛋白质查询序列与 DNA 数据库之间的比较，运行时作必要的转换。

另一个改进与初始计分的计算有关，在选出最佳区域后，FASTA 采取了一个额外步骤以合并邻近区域，即使这些区域不属于相同的对角线。这样，相关序列的初始计分能够得到相当的改进，更加接近于改进计分（FASTP 中的优化计分在 FASTA 中的说法）。进一步，相比于 FASTP 保持 5 个优化区域，FASTA 保持 10 个优化区域，该家族中的其他程序也使用相同的技术。LFASTA 用于局部相似性比对，它报告出两序列间多于一个的好比对。

统计工具也得到了改进，最突出的部分是局部重排。生物学上无太大意义的比对可

能因重排破坏了残基分布的不平均性而得到高 z 值。局部重排能够克服这个问题,它在10 到 20 个残基的块内进行,因此能够产生既随机又与原序列有相同局部组分的重排序列。如果高计分是由于倾斜的局部组分引起的,它将影响局部重排序列的平均计分。其他的改进包括选择计分矩阵的灵活性以及对每个重排序列计算更多的计分。

3.6 其他问题

我们现在研究与序列比较有关的几个杂项议题。第一个是两序列间的距离以及其与相似性的关系;第二个是做序列比较时我们的各种选择;最后我们简要讨论串匹配和确切序列比较。

*3.6.1 相似性与距离

相似性和距离是比较字符串的两个途径。使用相似性时,我们感兴趣的是两个串间的最佳比对,比对的计分给出两个串的相似性的度量。使用距离时,我们将不同的耗费值赋予基本编辑操作,并寻求一个开支小的方式把一个串转换成另一个串。因此距离是一个关于两序列差别的度量。在两种情形下我们均寻求一个数量值来度量序列差异或相似的度量。 90

到目前为止我们一直关注计算相似性。这个部分介绍距离及其与相似性的关系。我们将展示在许多情况下这两种度量可通过一个公式相互联系,因此可容易地从一个度量获得另一个度量。

我们必须强调对距离的研究仅限于全局比较中,它不适合局部比较。这一限制是我们在本章仍以相似性为基础的原因之一。表 3.2 概述了相似性与距离的主要差别,其中一些项读者可能还不熟悉,但我们接下来将会解释。

我们从精确定义相似性与距离的概念开始。令 s 和 t 是字母表 $\sum$ 上的两个序列,这个字母表通常是 DNA 或蛋白质字母表,但结论对其他一般情形也成立。我们是要寻找一个数字来度量串的相似或不同。和从前一样,我们用 $\mathrm{sim}(s,t)$ 和 $\mathrm{dist}(s,t)$ 分别表征相似性和距离。

表 3.2　相似性与距离性质概述

	相似性	距离
三角不等?	非	是
局部比较?	是	非
可能 $p(a,a)\neq p(b,b)$?	是	非

= = = = = = = = = = = 相　似　性 = = = = = = = = = = =

对相似性的度量总是基于比对。让我们回忆并精确定义比对。s 与 t 之间的比对是由对 s 和 t 插入空格符而得到的一个序列对 (s',t')。比对 $\alpha=(s',t')$ 必须满足:

1. $|s'|=|t'|$;
2. 移去 s' 中所有空格后得到 s;
3. 移去 t' 中所有空格后得到 t;
4. 对每个 i,或者 $s'[i]$ 或者 $t'[i]$ 须不是空格。

比对建立了符号 $s'[i]$ 与 $t'[i]$ 之间的联系，这两个符号在各自的序列中占据相同的位置 i。符号 $s'[i]$ 和 $t'[i]$ 称做在 α 下被比对。

相似性是任何比对的最大计分，且我们假定这个计分系统是累加性的。让我们同样回顾和精确定义什么是计分系统。一个计分系统是一个二元组 (p,g)，其成员是函数 $p:\Sigma\times\Sigma\mapsto\mathbf{R}$，用于对比对的符号对进行计分，另一个空格罚分函数 g 用于对空格进行罚分。通常 $g<0$，但我们不要求必须这样。用这样一个计分系统，我们能够对每个可能的
91 比对赋予一个数值(或计分)。在 α 中，每当 a 匹配于 b 时我们增加 $p(a,b)$，每当 a 与一个空格相配时增加 g，整个的和是比对 α 的计分。根据这个计分系统，两个序列间的相似性是：

$$\mathrm{sim}(s,t)=\max_{\alpha\in\mathcal{A}(s,t)} score(\alpha)$$

这里 $\mathcal{A}(s,t)$ 是 s 与 t 间所有比对的集合，这个计分系统称做是累加性的，因为如果我们把比对切成任意相继的两块，整个比对的计分是各块的计分的和。

＝＝＝＝＝＝＝＝＝＝＝距　离＝＝＝＝＝＝＝＝＝＝＝

集合 E 上的距离是一个函数 $d:E\times E\mapsto\mathbf{R}$，使得

1. 对所有 $x\in E, d(x,x)=0$；对 $x\neq y, d(x,y)>0$；
2. 对所有 $x,y\in E$ (d 是对称的)，$d(x,y)=d(y,x)$；
3. 对所有 $x,y,z\in E, d(x,y)\leqslant d(x,z)+d(y,z)$。

第三个条件称做三角不等性，这个性质在很多情况下都非常有用，许多算法依赖于它的有效性。对于串比对情形，根据把一个串转换到另一个串所需耗费的操作，我们可以在字母表 Σ 定义一个对所有串的距离。

通过连续施行一系列可允许操作，任何串都可被转换成其他串。如果我们对每个可允许操作赋予一个耗费(cost)，便能把两个串之间的距离定义为把一个串转换成另一个串的最小总耗费。可允许操作是：

1. 字符 a 替换为 b；
2. 插入或删除任意的空格。

为了对这些操作计价，我们使用一个耗费度量(cost measure)(c,h)，c 是一个函数 $c:\sum\times\sum\mapsto\mathbf{R}$，$h$ 是一个实数。一系列操作 σ 的耗费等于单个耗费的总和，用 $cost(\sigma)$ 表示。

根据耗费度量，两个序列之间的距离是：

$$\mathrm{dist}(s,t)=\min_{\sigma\in S(s,t)} cost(\sigma)$$

这里 $S(s,t)$ 是把 s 转换成 t 的所有操作的集合。

为了使这个定义更合理，有必要对 (c,h) 作某些限制。首先，我们仅处理非负耗费值，否则最小值将无意义。同时，假定耗费函数是一个对称函数，否则距离将不是对称的。这些最小需求保证 $\mathrm{dist}(s,t)$ 是对称的且满足三角不等性。为确保它是距离，我们还需要对 $a\neq b$ 有 $c(a,b)>0$ 及 $h>0$，以避免当 s 与 t 不相等时 $\mathrm{dist}(s,t)=0$。仅当序列相同
92 时才有距离等于 0。

我们还假设 c 满足三角不等性，即

$$c(x,y)\leqslant c(x,z)+c(y,z) \tag{3.15}$$

对于Σ中的所有 x、y 和 z 均成立。做出这一假设的理由是，即使我们开始时所用的度量 (c,h) 不满足这一不等性，我们总能定义一个新的能满足它的 (c',h) 并产生完全相同的距离函数。例如，如果有三个字符 x、y 和 z，使得 $c(x,y)>c(x,z)+c(z,y)$，则当我们要用 y 替换 x 时，我们不是直接操作而是用 z 替换 x 再用 y 替换 z，这样以更低的耗费产生相同的效果。对于实际情况，用 y 替换 x 的耗费仿佛是 $c(x,z)+c(z,y)$ 而不是 $c(x,y)$，因为实际开支是前者，而式(3.15)的规定则避免了这种情形[即 $c(x,y)$ 大于 $c(x,z)+c(z,y)$]。

例 3.3　令

$$c(x,y)=\begin{cases}0 & if\, x=y\\ 1 & if\, x\neq y\end{cases}$$

且 $h=1$，我们有所谓的两序列间的**编辑距离**(edit distance)，亦称做 Levenshtein 距离。

=========== 计 算 距 离 ===========

给定一个耗费度量 (c,h) 以及一个常数 M，我们能够定义一个计分系统 (p,g) 如下：

$$p(a,b)=M-c(a,b) \tag{3.16}$$

$$g=-h+M/2 \tag{3.17}$$

如果我们有一个序列 s 与 t 的比对 α，便可定义一个操作系列 σ 使得

$$score(\alpha)+cost(\sigma)=\frac{M}{2}(m+n) \tag{3.18}$$

这里 $m=|s|,n=|t|$。为得到 σ，只需如同先前计算计分时那样按列分割比对 α 即可，列与可允许操作相对应，字符匹配与替换相对应，空格与插入或删除相对应。操作可按任意次序进行，因为它们作用于比对的不相连区域，彼此并不相干。

让我们把从左到右施行的操作称做 σ。现在计算 α 的计分和 σ 的耗费。假定 α 中恰有 l 个字符匹配，第 i 个匹配是 s 中的 a_i 和 t 中的 b_i，进一步假定 α 中恰有 r 个空格。基于这些，我们有如下的关于 α 的计分的表达式：

$$score(\alpha)=\sum_{i=1}^{l}p(a_i,b_i)+rg$$ 93

另一方面，σ 的开支是

$$cost(\sigma)=\sum_{i=1}^{l}c(a_i,b_i)+rh$$

把这两个等式两边相加，根据式(3.16)和(3.17)，我们有

$$score(\alpha)+cost(\sigma)=lM+r\frac{M}{2} \tag{3.19}$$

注意 l 和 r 的值不是独立的，它们必须满足一个与两序列字符总数有关的一个关系。因每个匹配涉及两个字符，一个空格仅涉及一个字符，因此字符总数必须是：

$$m+n=2l+r \tag{3.20}$$

由此，我们可把式(3.19)写成

$$score(\alpha) + cost(\sigma) = \frac{M}{2}(m + n)$$

而它恰好就是(3.18)

该结论对任何比对都成立。特别是当 α 是一个优化比对时,我们有

$$\text{sim}(s, t) + cost(\sigma) = \frac{M}{2}(m + n)$$

这里 σ 代表由 α 导出的操作系列。由于 s 与 t 之间的距离是任意操作系列的最小耗费,我们最终有

$$\text{sim}(s, t) + \text{dist}(s, t) \leqslant \frac{M}{2}(m + n)$$

现在我们用定理概述以上结果以便以后引用。

定理 3.2 给定一个耗费度量(c, h),一个 M 值,以及一个计分系统,对任何串 s 和 t 我们有:

$$\text{dist}(s, t) + \text{sim}(s, t) \leqslant \frac{M}{2}(|s| + |t|)$$

定理 3.2 给出了一个 $\text{dist}(s, t)$的上界值,我们现在要证明它同时也是一个下界值,且距离可直接由相似性获得。

定理 3.3 根据定理 3.2 的条件,我们有

$$\text{sim}(s, t) + \text{dist}(s, t) \geqslant \frac{M}{2}(|s| + |t|)$$

证明:我们从定理 3.2 入手。首先,对于一个可接受操作系列 σ 我们构造一个带下述性质的比对 α

94

$$score(\alpha) + cost(\sigma) \geqslant \frac{M}{2}(m + n) \tag{3.21}$$

注意,我们这里不能有如同前面用到的一般性的相等性。这个例子有些不同,因为施行了一个又一个的操作后对应的比对仅显示净效应。例如,如果 σ 在某些点插入一个字符 a 然后又很快删除它,其净效应是空操作,两个操作的耗费相同。前述的方程必须涵盖所有的操作,不论它们是否显效。这就是为什么我们这里用不等式而不是等式。

我们通过对 σ 中操作的数量进行归纳来证明式(3.21)。如果 $|\sigma| = 0$,没有操作发生,因此 $s = t$。比对(s, s)因此有计分 $Mm = Mn = M(m + n)/2$,等式此时成立。

如果 σ 至少有一个操作,考虑它的最后一个操作,我们可把 σ 写成 $\sigma = \sigma' u$,u 是最后一个操作,因此:

$$s \xrightarrow{\sigma'} t' \xrightarrow{u} t$$

即 t'是通过对 s 施行除最后一个操作以外所有其他操作获得的串,t'是通过对 t 施行 u 操作获得的串。

我们的归纳假设是,在 s 与 t'之间我们有一个比对 α'使得

$$score(\alpha') + cost(\sigma') \geqslant \frac{M}{2}(m + n')$$

这里 $n' = |t'|$。因为 $cost(\sigma) = cost(\sigma') + cost(u)$,我们需要做的是找出 s 与 t 之间的一

个比对 α 使得

$$score(\alpha) \geqslant score(\alpha') + \frac{M}{2}(n - n') - cost(u) \tag{3.22}$$

把这两个等式两边相加便得到我们所需的结果。根据 u 操作的类型，有三种情形需要考虑：替换、插入、删除。

情形 1： u 是一个替换

这时 $cost(u) = c(a,b)$，$n = n'$，t 与 t' 仅差一个位置

$$\begin{array}{ll} s & \dots x \dots \\ t' & \dots a \dots \\ t & \dots b \dots \end{array}$$

通过用 α' 中的 b 替换 a，我们便构造了 α。如果 s 中的符号 x 是一个普通符号，则

$$\begin{aligned} score(\alpha) &= score(\alpha') - p(a,x) + p(b,x) \\ &= score(\alpha') - M + c(a,x) + M - c(b,x) \\ &= score(\alpha') + c(a,x) - c(b,x) \\ &\geqslant score(\alpha') - c(a,b) \end{aligned}$$

这里最后一步是引用 c 的三角不等式的结果。因此在此例中我们得到了所需的结果。 94

如果 x 是一个空格，α 和 α' 的计分便相等，因此

$$score(\alpha) = score(\alpha') \geqslant score(\alpha') - c(a,b)$$

因为根据假设有 $c(a,b) \geqslant 0$。

情形 2： u 是一个插入

这时 $cost(u) = h$，且 $n' = n - 1$。比对 α' 在某个位置必须是“开”的，以接纳某个字符的插入，如下所示

$$\begin{array}{ll|l} s & \dots & \dots \\ t' & \dots & \dots \end{array} \qquad \begin{array}{ll|c|l} s & \dots & - & \dots \\ t & \dots & a & \dots \end{array}$$

一个额外空格被加入 s。α 中的这个额外空格导致 α 与 α' 的计分差别：

$$\begin{aligned} score(\alpha) &= score(\alpha') + g \\ &= score(\alpha') - h + \frac{M}{2} \end{aligned}$$

这证明了当 u 是插入操作时式(3.22)的有效性。

情形 3： u 是删除

此时仍有 $cost(u) = h$，但 $n' = n + 1$，我们有如下情形

$$\begin{array}{ll} s & \dots x \dots \\ t' & \dots a \dots \\ t & \end{array}$$

根据 x 是否为空格，我们需区别两种情况。如果 x 是一个字符，则

$$\begin{aligned} score(\alpha) &= score(\alpha') - p(a,x) + g \\ &= score(\alpha') - M + c(a,x) - h + \frac{M}{2} \end{aligned}$$

$$= score(\alpha') - h - \frac{M}{2} + c(a,x)$$

$$\geqslant score(\alpha') - h - \frac{M}{2}$$

因为 $c(a,x) \geqslant 0$。如果 x 是一个空格,我们有

$$score(\alpha) = score(\alpha') - g$$

$$= score(\alpha') - h + \frac{M}{2}$$

$$\geqslant score(\alpha') - h - \frac{M}{2}$$

因为 $h > 0$。

由此完成归纳证明,式(3.21)成立。特别是,它对最小耗费 σ 成立,使得

96 $$score(\alpha) + \mathrm{dist}(s,t) \geqslant \frac{M}{2}(m+n)$$

或者是,考虑到 $\mathrm{sim}(s,t)$是 s 和 t 间的最大可能比对计分,我们有

$$\mathrm{sim}(s,t) + \mathrm{dist}(s,t) \geqslant \frac{M}{2}(m+n)$$

从而证明了定理。 ■

我们可把定理 3.2 和 3.3 写成

$$\mathrm{sim}(s,t) + \mathrm{dist}(s,t) = \frac{M}{2}(m+n) \tag{3.23}$$

它告诉我们如何由相似性计算距离,因此距离计算可还原为相似性计算。要计算一个距离,所需做的是选择一个合适的 M,按式(3.16)和(3.17)定义计分参数 p 和 g,并使用一个全局比较算法,所得到的相似性可按上式转换为距离。

例如,在编辑距离的例子里我们可选择 $M=0$,并运行一个相似性算法,规定匹配为 0 分,失配为 -1 分,空格为 -1 分。也可以选择 $M=2$,规定匹配为 2 分,失配为 1 分,空格为 0 分。虽然分值不同,两个计分系统产生的优化比对却是相同的,且应用式(3.23)后距离也是相同的。

结束本部分之前有必要解释一下 M。对任何 M 值都适用于定理 3.2 这一点,读者看起来可能有些怀疑。如果 M 是一个大的正数,对空格罚分函数 $w'(k)$中的许多甚至所有 k 值计分系统可能产生负值,这与我们的直觉相悖,因为空格应被减罚而不是褒加。当 M 是一个绝对值大的负数时情形类似,这时函数 p 将是负的,亦与直觉相悖。这种表面异常的原因是我们仅考虑全局比较。当我们改变 M 值时,所有的比对按相同方式增加或减少它们的计分,使得优化比对总保持不变。对于局部比对,等式(3.20)不再有效,改变 M 值使得我们可选择更长或更短的局部比对。

3.6.2 序列比较中的参数选择

这一部分我们介绍计分系统中参数选择的若干考虑,以及针对特定序列比较的算法选择。

在选择计分系统时要考虑到许多因素,至少包括匹配分值、失配分值、空格分值。

在任何计分系统中都要保证匹配分值高于失配分值，从而有利于相同字符的比对。另一条常常使用的规则是，失配比使用一对空格为好。例如，下列的最左比对应比最右比对有更高的得分： 97

A　-A
C　C-

以上规则可翻译成包含计分参数的不等式：

$$2g < m < M$$

注意，如果我们把所有的权值乘以一个正整数，优化比对仍然不变，这一性质可用于将所有权值转换为整数，而在大多数计算机里整数的运算速度是大大快于浮点数的。

选择考虑下列的变换。如果我们有序列 AT 和 TA，则存在两个最佳计分：

AT　-AT
TA　TA-

（对上述第二个比对，还存在着有同样计分的第三个比对，对齐 T）对应的计分分别是 $2m$ 和 $M+2g$。因此如果 m 等于 M 和 $2g$ 的算术平均值，这两个比对就在计分上等价。为了确定优先者，我们必须选择 m 的值，使其靠近区间 $[2g, M]$ 的某一端。

同样地我们可以想像其他的短序列比对情形，考虑哪种情形最切需要，而对 m、M、g 亦有更多的不等关系。例如，当比对序列 ATCG 和 TCGA 时我们可能偏向下面两个中的前者：

ATCG-　ATCG
-TCGA　TCGA

如果 $4m < 3M + 2g$ 的话，差别将会在计分中反映出来。3.2.1 中所用到的值即按此标准确定。

在实际中，我们需要更复杂的计分系统，而不是仅 M、m、g 这样简单的系统。3.3.3 中提到的次加性(subadditive)空格罚分函数通常更为合用。其中，仿射函数(affine function)十分流行，因为它有平方性运行时间而非一般函数的立方性运行时间。另一个重要的性质是区别各种不同氨基酸匹配的能力（对于蛋白质比较），具有相似化学或物理性质（诸如大小、电荷、亲水性等等）的氨基酸之间的匹配相比于性质不同氨基酸之间的比对具有更高的分值。在这种情况下，仅仅基于符号一致的计分系统就不够了。蛋白质比较中常用的是 PAM 矩阵。

不过，简单的符号一致/不一致方法十分通用，足以处理序列比较中一些熟知的问题，其中一例便是发现序列对的*最长公共子序列*(longest common subsequence, LCS)，这等价于在基本算法中规定 $M = 1$、$m = g = 0$。因此，我们能够用 $O(mn)$ 的时间和 $O[\min(m, n)]$ 的空间求解这个问题，m、n 分别是两个序列的长度。这个问题受到大量的关注，对几个特例已存在更快的算法。

98

我们现在讨论算法的选择。对于是否对终端空格罚分以及使用局部或全局比对这样的问题，答案取决于应用的类型以及获求的结果。如果要比对的序列长度近似且序列相近，则对所有空格进行罚分的全局比较较为合适，例如两个不同物种 tRNA 的比较或相同物种但携带氨基酸不同的 tRNA 的比较。

另一方面，如果一个序列短另一个序列长，只对较短序列的终端空格进行罚分更为可取，这使得我们可以找出短序列在长序列中的所有近似出现。当试图在新测序的 DNA 中发现已知的保守结构时这种比较十分有用。

当我们要比较两个长的可能包含高度相似区域的序列时，则宜用局部比较。一个典型例子是具有相似功能的蛋白质序列，它们来自进化中高度分离的物种。因为蛋白质具有相似功能，很有可能两个蛋白质具有高度相似的区域[活性位点，基序(motif)，功能等价结构等等]，而这些区域被无关区域分隔。突变多在无关区域累积，且对蛋白质功能没有大的影响。

如果序列比较是为了探知它们是否有公共起源，对结果的解释就必须十分小心了。一般从概率的角度说优化比对是不太可能偶然出现的。然而我们还是建议将所获得的计分与完全无关序列的平均比对计分相比较，且这些无关序列应具有与比对序列类似的性质。如果优化计分显著高于平均值，则表明序列间的相似性并非偶然。即使这样，这个结果本身也并不意味着序列是同源序列或有共同起源，尚需要根据比对结果作进一步的实验，以强化或否决共同起源的假设。另一方面，如果相似性几近与偶然情况相等，则序列很可能是无关的。但是，在生物学里没有无例外的规则，缺乏序列层次上相似性指征的同源蛋白是存在的，在这种情况下其他的证据，如三维结构，可用于确定同源性。

3.6.3 串匹配和确切序列比较

另外两个与计算分子生物学有关的问题是串匹配问题和确切序列比较。在串匹配问题里我们有一个串 s，$|s| = n$，以及另一个串 t，$|t| = m$，我们要找出 t 在 s 中的所有出现。换言之，即 t 是否为 s 的一个子串。如果是，哪些是 t 在 s 中的所有出现位置？这是一个存在高效解法的经典计算机科学问题。实际上，存在 $O(m+n)$时间的算法，它大大
99 快于两序列比较基本算法的平方复杂性。我们这里不讨论这类算法，但读者可在任何好的算法教科书中找到，包括在第二章中文献提要部分给出的书。我们注意到对近似串匹配问题亦发展了高效算法。“近似”指的是寻找匹配时允许有一定数量的误差，这种算法在分子生物学中当然是重要的，参考文献在文献提要中给出。

本章的大部分都是关于序列比较的，这种比较允许序列中存在错误，绝大多数实际应用均是此种情形。但有时我们亦需要*确切比较*(exact comparison)，许多这类问题可由后缀树求解，它是一种非常有用且适用面广的结构。一些例子是：

- 我们将在第 4 章中看到，在大规模 DNA 测序时一个序列中的重复子串是一个严重的问题。给定一个 DNA 序列，一个自然的问题是找出它的最长重复子串。
- 当使用聚合酶链反应时(在 1.5.2 描述)，我们需要找出一个不在需复制的 DNA 中的短的 DNA 片段，称做引物。这个问题可形式化为：给定两个串 A(长)和 B(短)，对 B 中的每个位置 i，找出一个起始于 i 的最短串，它不在 A 中出现。
- 在 1.5.2 中看到，限制酶的限制位点是偶数长度的回文。另一个问题是对给定 DNA 找出所有最大回文。

我们现在描述什么是后缀树。它基本上包括一个串的所有后缀，尽可能排除了树结构中的公共前缀。更形式地，$s = s_1 s_2 \ldots s_n$ 的一个后缀树是一个有 $n+1$ 个叶子的有根树

T,它有如下性质:

- T 的边的方向离开根,且每边用 s 中的一个子串标记。
- 所有发自一个顶点的边有不同的标记,所有这样的标记有不同的前缀(不记空前缀)。
- 每个叶子对应于 s 中的一个后缀,这个后缀由所有从根到该叶子的路径上边的标记拼接而成。

图 3.19 为后缀树的一个例子。为了避免空后缀问题,我们通常在 s 尾部添加一个额外字符 \$ 。后缀树能够用漂亮的算法构造,运行时间为 $O(n)$,假定字母表的大小是常数。我们这里不拟介绍该算法。一旦构造了后缀树,大多数相关问题可在时间 $O(n)$内求解,上述例子即是如此。引物问题可在时间 $O(|B|)$内求解。我们把如何应用后缀树求解该问题作为习题留给读者。

一个叫做后缀阵列的相关结构也能用于确切比较问题,这个结构基本上由 s 的所有 100
按字母顺序排序的后缀组成。

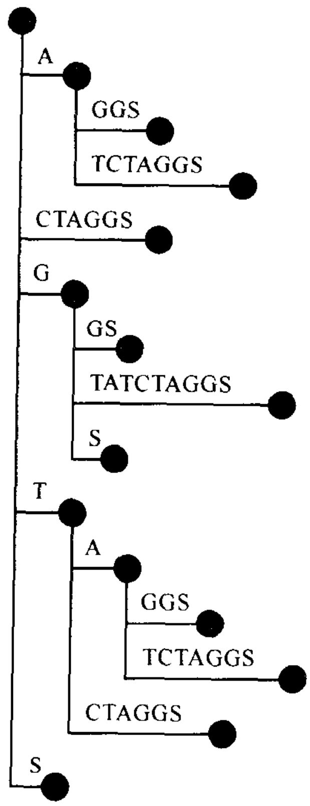


图 3.19　串 GTATCTAGG 的后缀树,一个美元符号加在串尾

小　　结

我们从比较两个序列开始。一个称做动态程序设计的算法设计技术可成功地应用于

这种问题，产生精确和有效的算法。我们接着研究了基本动态程序设计算法的几个扩展，一些扩展的算法使得我们能够在作某些计算时节省空间或时间。另一些扩展则着眼于求解更一般的生物学问题。

下面是各种序列比较类型及其用途的简要回顾：

- **全局**　比较整个序列
- **局部**　发现序列中的相似片段
- **半全局**　发现后缀－前缀或包容相似性

所有这些算法在时间和空间上都具有平方复杂性。然而，也存在节省空间的版本，这
101 些版本使用线性的空间，但要耗费双倍的时间。

下面的表格根据空隙罚分函数的类型总结了不同算法的性质。

函数类型	表达式	复杂性	所用空间
线性	$w(k)=ak$	$O(n^2)$	一个阵列
仿射	$w(k)=ak+b$	$O(n^2)$	三个阵列
一般函数	任何 $w(k)$	$O(n^3)$	三个阵列

接下来我们研究了多序列比较。这时，基于动态程序设计产生的算法是低效的，因此必须考虑其他技术。我们描述了许多候选技术。动态程序设计不能胜任的另一个情形是对给定的序列或模式搜索数据库。我们在 3.5 节中研究了这个问题以及数据库搜索软件中的启发式算法。

本章的大部分内容都基于相似性概念，但这不是作序列比较的惟一途径。距离的概念也同样重要，用于度量两序列的差异程度。在 3.6.1 中我们研究了它们两者间的关系。我们以一个简要讨论结束本章，概述比较方式与参数选择中的若干问题。

习　　题

1. 找出下列的算法与问题之间的对应性
   半全局比较
   过滤(filter)并应用于半全局比较
   K 带
   数据库搜索
   局部比较
2. 根据下列标准对 3.2 节的比对(3.1)进行计分：如果 $a=b$ 则 $p(a,b)=1$；如果 $a\neq b$ 且 $g=-1$ 则 $p(a,b)=0$
3. 用 3.2.1 中的计分系统找出 AAAG 和 ACG 之间的所有优化比对。
4. 虽然最上比对和最下比对经常包含两序列间的所有优化比对，但情况并不总是如此。找出一个例子。
5. 考虑习题 2 和 3.2.1 中的计分系统。找出两个序列，其优化比对在两个计分系统下不同。

6. 修改图 3.3 中的算法 *Align*,使得它能发现 s 与 t 间的一个优化半全局比对,对 s 与 t 端点的空格不罚分,且阵列 a 已被填充。 102
7. 找出一个 ATACTACGGAGGG 和 GAACGTAGGCGTAT 之间的最佳局部比对,使用你最喜欢的计分系统。
8. 两个序列的局部相似性与全局相似性哪个值更大? 为什么? 它们的半全局相似性如何与另两个值相比较?

*9. 提出局部与半全局相似性的线性空间算法,以及优化比对的构造。

10. 确定下面陈述的正确性,并检验你的答案:s 与 t 间的两个优化比对将有同样数量的块(参见 3.3.2 中对比对块的定义)。
11. 回忆 3.3.4 中的概念。找出一对序列使得 $k=0$ 的最佳计分与 $k=1$ 的最佳计分相同,但一定小于 $k=2$ 的最佳计分。本问题中 $M=1, m=g=-1$。
12. 修改算法 *KBand*(图 3.9)使得它能处理不等长序列。修改后算法的时间复杂性如何?
13. 使用字符对比较的和计算多序列比对,找出 ATC,CAGC,CGC 间的所有优化多序列比对,如果 a、b 均为字符且 $a=b$,则 $p(a,b)=1$;如果 a、b 均为字符且 $a\neq b$,则 $p(a,b)=0$;$p(a,-)=-1$。
14. 在图 3.14 的多序列比对算法中,除了相关单元池,还需要多少空间?
15. 证明 k-PAM 计分矩阵对每个 k 都是对称的。
16. 1-PAM 矩阵平均改变 1% 的氨基酸。我们能否说 2-PAM 矩阵平均改变 2%?
17. 用 BLAST 在 GenBank 中查询序列 GAATTCCAATAGA,命中了什么数据库序列?
18. 一个空隙减罚函数是凹的,当 $\triangle w(k) \geqslant \triangle w(k+1)$。证明一个 $w(0)=0$ 的凹函数是次加性的(见 $\triangle w(k)$ 的定义)。

*19. 真或假:一个次加性和非降的空隙减罚函数 w 必然是凹的。

20. 考虑序列 s 和 t 间的一个长为 l 的比对。把该比对拆成两部分,一个从列 1 到列 k,另一个从列 $k+1$ 到列 l。计分系统的什么性质保证对任意的 k 整个比对的计分是其部分比对计分的和?
21. 例 3.2 中的指派是否是优化的?
22. 根据相似性与距离之间的关系,给出一个对应于编辑距离的计分系统。

◇23. 给定两个长度分别为 m 和 n 的 DNA 序列,找出一个它们间的全局相似性分布。当 m 和 n 增长时该分布是否逼近任何已知的分布?

24. 设计一个算法发现两序列间的最短公共超串。提示:可通过对基本序列比较算法的修改而得到。
25. 如何使用后缀树解决 3.6.3 中提到的问题。 103

文 献 提 要

Sankoff 和 Kruskal 的书是关于序列比较的经典参考书[168]。Waterman 对优化比对、相似性、距离以及相关算法作了综述[196]。Pearson 和 Miller 的文章则着重于动态程序设

计方法[156]。Heijne 的书则是有关序列比较的另一个有用的资料来源[191]。

尽管 S. Ulam 在 20 世纪 50 年代就已经考虑了序列空间距离，从生物学家的角度看，Needleman 和 Wunsch 的名篇被广泛认为是对序列比较的第一个重要贡献[146]。Needleman 和 Wunsch 的算法对空隙进行了固定的减罚，与长度无关，文中也未对复杂性进行分析。后来的分析表明他们的算法是立方复杂性的。现在我们知道如何在平方复杂性时间里使用仿射空格减罚函数执行该算法。在任何情形下，Needleman 和 Wunsch 算法即意味着任何基于动态程序设计技术的全局比对算法。

局部比对也有一个类似现象。Smith 与 Waterman 的文章影响甚深[175]，导致他们的名字成为几乎所有基于动态程序设计的局部比对的代名词。文章本身非常短，仅 3 页，提供了一个填写阵列的公式，不带复杂性分析。文中所示的例子使用了一个仿射空格罚分函数。后来，Huang 及其合作者发展了该算法的平方复杂性线性空间版本[100]。Waterman 描述了能够得到接近优化比对的版本[197]。

现在仍不清楚 3.2.1 中的基本算法何时首次出现，也许是在不同的情形下被多次发现[172]。有用的空间节省技巧由 Hirschberg 给出[94]，它后来被扩展成动态程序设计序列比较算法的最重要版本[100,145]。

文献中大量篇幅被用于凹的空格减罚函数[140]，部分是因为它们能够很好地实现分离的空格减罚更多这一概念，正如次加性函数那样。凹性和次加性通常是不可互比的，尽管凹性加上额外条件 $2w(1) \geqslant w(2)$ 意味着次加性(见习题 18)。

本书中的相似序列比较主要基于 Fickett[63] 的文章。Ukkonen[188]、Myers[42] 以及 Chang 和 Lawler[34] 发展了更有效的算法用于具有特殊性质的计分机制。

PAM 矩阵由 Margareth Dayhoff 及合作者引进[45]，有文献对其进行了介绍[70]。Altschul[10] 给出了一个合理的分析，表明 PAM 距离应根据特定的比较进行选择。Karlin 和 Altschul[109]、Karlin、Dembo 和 Kawabata[110] 探讨了有关的统计学意义，它们对选择 PAM 距离有重要影响。对搜索序列数据库的许多相关问题可见 Altschul 等的论著[11]。有别于 PAM 矩阵的另一个方法是 Henikoff 和 Henikoff[93] 提出的，称做 BLOSSUM 矩阵。

104 研究者们还考察了同时用多种计分计算优化比对的方法，见 Waterman[198]。

Lipman 和 Pearson[127]、Pearson 和 Lipman[155]、Pearson[153] 描述了 FAST 的实现。Pearson[154] 彻底比较了 FASTA 与纯动态程序设计。BLAST 程序是 Altschul 等人[12] 描述的。如何快速发现公共区以便构造更大的比对是一个一再提及的论题。在各种应用中，我们推荐 Chao、Zhang、Ostell 和 Miller[36] 的极端长序列的局部比对，亦可参见 Chao 和 Miller[35]，以及 Joseph、Meidanis 和 Tiwari[104] 的文章。

有许多文献讨论距离与相似性之间的关系。3.6.1 中的标准转换是 Smith、Waterman、Fitch[176] 提出的。Waterman 也谈了这一问题[197]。也有文献给出了一个允许空间一空间匹配的比对算法[134]。

在多序列比对中用于节省时间的投影方法由 Carrillo 和 Lipman[32] 给出。Altschul 和 Lipman[13] 把它一般化，应用到星比对。Gusfield 也讨论了星比对[83]，证明了符号配对和(sum-of-pairs)距离度量方法能够有性能上的保证。Gupta、Kececioglu 和 Schäffer[81] 实现了 Carrillo 和 Lipman 的算法，即多序列比对程序 MSA。Kececioglu 对"与一个配对比对集

合尽可能接近的多序列比对"的概念进行了形式化[113],把它称做最大权踪迹问题(maximum weight trace problem)。他证明该问题是 NP-难题并给出了一个分枝与界限算法。还有另一种基于踪迹(trace)概念的序列比对启发式算法[137]。带 SP 度量的多序列比对和树比对均是 NP-难题,见 Wang 和 Jiang[193]。树比对的指数时间算法见 Sankoff[166]。基于距离的树比对的近似算法由 Wang 和 Gusfield[192]提出,改进了较早的 Jiang、Lawler、Wang[102]的结果。某些多序列比对的启发式必须解决对齐比对问题,两个这方面的文献是 Miller[139]和 Gotoh[77]的文章。有关多序列比较不同方法的回顾见 Chan、Wang、Chiu[33]。

因为严格的最佳比对方法实际运行非常慢,并行计算机的使用常常成为提速手段。Jones 等人的文章[103]讨论了这一问题。

后缀树和关于串的许多其他算法在 Crochemore 和 Rytter 的书[39]以及 Stephen 的书[180]中描述。后缀阵列出现于 Manber 和 Myers 的论文[130]。3.6.3 中提及的后缀树在引物选择问题和回文问题中的应用由 Dan Gusfield 给出。Gusfield 的一部新书将详细描述许多串算法以及它们在计算生物学中的应用[85]。

第 4 章 DNA 片段组装

在第一章我们涉及了 DNA 测序的生物学方面，本章我们讨论与测序有关的一个计算问题，即片段组装。这个问题源自这样一个事实，即使用现有技术不可能直接连续测序超过几百个碱基的片段。另一方面，现有技术可以把长 DNA 分子随机切断，产生足够的小片段副本以供测序。因此，测序长 DNA 分子的一个典型途径是测序所采集样本的小片段。然而，片段组装则是由此引发的一个问题，这正是本章要讨论的。我们将介绍问题的形式模型和求解的算法。

4.1 生物学背景

105 **测序**(sequence)DNA 分子就是获知其所含的碱基串。在大规模测序中，待测序的目标 DNA 分子通常长达几千碱基对。人们可把这个问题看做猜字字谜，有两行面向桌面的纸牌，我们不知道每张纸牌上写的是{A,C,G,T}中的哪个字母，如图 4.1 所示，但我们知道某行某位置纸牌的字母与对应行对应位置纸牌的字母互补。我们的目标是根据线索判读字母，所要构造的序列称为**目标序列**(target)。

5′ ···□□□□□□□□··· 3′
3′ ···□□□□□□□□··· 5′

图 4.1 待测序的未知 DNA

在这个生物学问题里，目标序列的大致长度是知道的(误差 10%左右)，但不能直接测序整个分子。我们可以得到 DNA 分子某个链上的一个随机碎片，按标准的 5′→3′方向
106 测序，每个这样的小序列称为一个**片段**(fragment)，它是目标分子某个链的子串(但我们不知道是哪一链以及其相对于起始点的位置，而测序也可能包含错误)。使用霰弹枪法我们可获得并测序大量的片段，然后根据片段间的重叠来重构目标分子。根据实验条件，片段长度可以在 200～700bp 之间。对一个典型的测序问题，目标分子的长度为30 000～100 000bp，而片段的数量是500～2000。

这个问题(目标分子测序)由此变为根据测序的小序列推断目标 DNA 分子的完整序列。鉴于我们已经有测序的片段，这个任务因而被称为**片段组装**(fragment assembly)。原始分子的另一链亦能确定，因为双链的碱基对是互补的。

在余下部分我们给出该问题生物学性质的其他细节，它们对于片段组装算法的设计

是重要的。

4.1.1 理想情形

给出一个例子是研究相关问题的好方法，但注意真正的实例远比这里展示的例子要大。假定输入是以下 4 个序列片段

ACCGT
CGTGC
TTAC
TACCGT

且我们已知答案是一个 10 碱基对左右的序列。组装这个序列集的一个可能方式是

```
--ACCGT--
----CGTGC
TTAC-----
-TACCGT--
_________
TTACCGTGC
```

注意我们对齐了输入片段，忽略了两端的空格。我们试图把碱基对相等的列对齐。除了目标序列的大小，组装的惟一根据是片段间的交叠(overlap)，即某个片段的结尾与另一个片段的开头相似。片段 1 与 2 就存在这种情形。通过调整片段位置，我们得到一个**排列**(layout)，它可看做是片段间的一个多序列比对。 107

横线下的序列称**表决序列**(consensus sequence)，或简称*表决*(consensus)，它是我们的问题的答案。表决是由一列中所有碱基的多数表决机制决定的。在本例中，每一列碱基都是完全一致的，因此计算也直截了当。这个答案包含 9 个碱基，接近假定的 10 个碱基的目标序列，每个片段恰是一个子串。然而实际上片段难得恰好是表决序列的子串。

4.1.2 复杂情形

如上所述，实际问题的规模是非常大的，此外，其他几个因素也增加了问题的复杂性。其中最主要的因素包括错误、未知的序列朝向、重复区域、缺乏覆盖(coverage)。下面逐个描述这些因素。

===========错　误===========

最简单的错误称做*碱基识别错误*(base call error)，由片段中碱基的替换、插入、删除组成。相应的例子见图 4.2、图 4.3、图 4.4。此外，碱基转位也常见，但我们可用一个插入和一个删除(或两个替换)来描述。

在实际中，碱基识别错误的出现率约为每百字符 1～5 个，在序列中的分布也不是均衡的，而是趋向聚积于片段的 3′端。从例子可看到，即便有错误我们仍可重构出正确的表决，但计算程序必须能处理这种错误，而这通常意味着算法需要更多的时间和空间。例如，有可能在线性时间里发现两个没有错误的序列的最佳比对，而从第 3 章我们知道处理

```
输入：          答案：

ACCGT           --ACCGT--
CGTGC           ----CGTGC
TTAC            TTAC-----
TGCCGT          -TGCCGT--
                ---------
                TTACCGTGC
```

图 4.2　在本例中，最后片段的第二个位置有一个替换错误，A 被替换成 G。由于多数表决机制，表决仍然是正确的

```
输入：          答案：

ACCGT           --ACC-GT--
CAGTGC          ----GAGTGC
TTAC            TTAC------
TACCGT          -TACC-GT--
                ----------
                TTACC-GTGC
```

图 4.3　在本例中，第二个片段的第二个位置有一个插入错误，即碱基 A 原本是不存在的。表决仍然正确，因为在多序列比对中引入了空格以及多数表决机制。注意空格在汇结答案时是被忽略的

```
输入：          答案：

ACCGT           --ACCGT--
CGTGC           ----CGTGC
TTAC            TTAC-----
TACGT           -TAC-GT--
                ---------
                TTACCGTGC
```

图 4.4　在本例中，最后一个片段的第三(或第四)位置有一个删除错误。序列中引入的空格以及多数表决机制使表决仍然正确

带空格的序列则需要平方时间的算法。

除了错误的碱基识别，另两类错误也影响组装，一类是人为嵌合片段(artifact of
108 chimeric fragment)，另一类是宿主或载体(vector)DNA 污染。

当来自目标分子不同部分的两个正常片段端-端相连时，便产生一个不连续的片段，称为嵌合片段或嵌合片(chimera)。图 4.5 是一个例子。这些错误的片段必须被识别并在预处理阶段将它们清除。

```
输入：          答案：

ACCGT           --ACCGT--
CGTGC           ----CGTGC
TTAC            TTAC-----
TACCGT          -TACCGT--
TTATGC          ---------
                TTACCGTGC

                TTA---TGC
```

图 4.5　在本输入集合中最后一个片段是嵌合片。处理嵌合片的惟一办法是识别并在组装开始之前将它们移出输入集合

有时输入集合中也出现与目标分子无关的片段或部分片段，这由宿主或载体分子的污染所致。我们在 1.5.2 中已知，片段的复制过程包括将它们插入载体 DNA，并使携带片段的载体 DNA 不断复制。最后，片段必须从载体 DNA 中纯化出来，而纯化不完全正是污染的起因。如果载体是病毒，则被感染的细胞(通常是细菌)亦可能掺入某些遗传物质到复制的片段。

鉴于载体 DNA 数量相当大，测序中污染十分常见，使得科学家有时未能在组装前滤除载体序列从而向数据库提交了带污染的表决序列。

对于嵌合片，一个处理对策是在组装前过滤数据。DNA 测序中用到的载体的完整 109
DNA 序列是已知的，因此从中将污染片段筛除出来并不困难。嵌合片段检测导致有趣的算法问题，但我们不拟在本书中详述，有关文献见文献提要。

=========== 未知的序列朝向 ===========

片段可能来自 DNA 的任一单链，通常我们无从知道它属于哪一链。然而，我们知道它总是从 5′方向向 3′方向阅读。由于双链的互补性和反向性，一个片段是某链的子串等价于它的逆的补串是另一链的子串，因而我们可以把输入片段看做是表决的子串。图 4.6 显示了一个包含双向片段的组装问题，片段是未知的，但不包含错误。实际上，我们必须同时处理两种错误和两种未知朝向。

输入：		答案：
CACGT	→	CACGT--------
ACGT	→	-ACGT--------
ACTACG	←	--CGTAGT-----
GTACT	←	-----AGTAC---
ACTGA	→	--------ACTGA
CTGA	→	---------CTGA
		CACGTAGTACTGA

图 4.6 未知朝向的片段组装。初始时我们不知片段的朝向，每个片段可按正向或反向拼接。最后，我们用箭头指示所选的方向，→指原本方向，←指反转方向

由于朝向是未知的，我们实际上必须测试各种组合，而对 n 个片段存在 2^n 种朝向的 110
组合。当然这个方法是不现实的，也不是在组装中用到的方法，但它展示了朝向问题的复杂性。

=========== 重 复 区 域 ===========

重复区域或**重复段**(repeat)是在目标分子中出现两次或多次的序列，图 4.7 给出了一个例子。短的重复段，即能被单个片段完全覆盖的重复段，处理起来并不困难，糟糕的是长的重复段。此外，一个重复段的不同拷贝也可能不完全一致，亦给组装带来麻烦。如果一个重复段的两个拷贝相似度很高，则它们的差别可能由于碱基识别错误而被混淆。由于组装程序需能处理错误，因此对交叠的检测便存在一定的误差。

重复段产生两方面的问题。首先，如果一个片段完全包含于一个重复段中，则在目标序列中就可有几处位置来排放这个片段，因为片段在几个重复段拷贝中均可很好地吻合。

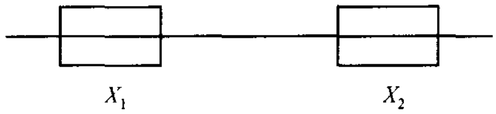


图 4.7　重复区域，X_1 和 X_2 是近乎相同的序列

有人可能会说放在哪里都没关系，因为不管在重复段的哪个拷贝最终产生的表决都相差不大。但问题是，当拷贝不确切等同时，排于错误的重复段拷贝中的片段可能使表决的质量下降。

其次，重复段的某些排列也可能使组装产生内在的歧义，即两个或多个的排列均与输入片段兼容，目标序列的长度也相当。图 4.8 和图 4.9 是两个这样的例子。第一个例子含同一重复段的三个拷贝，第二个例子含不同重复段的两个交织拷贝。这两个例子的共有特征是，两个不同区域夹在相同的重复段中。在第一个例子中，B 和 C 都夹在两侧的 X 中，在第二个例子中，B 和 D 都夹在两侧的 X 和 Y 中。

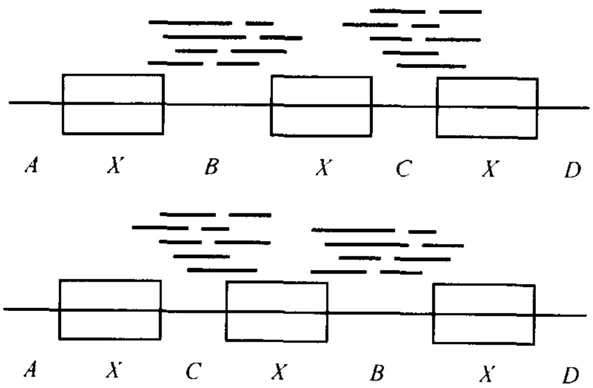


图 4.8　由 XXX 型的重复导致的目标序列的歧义性组装

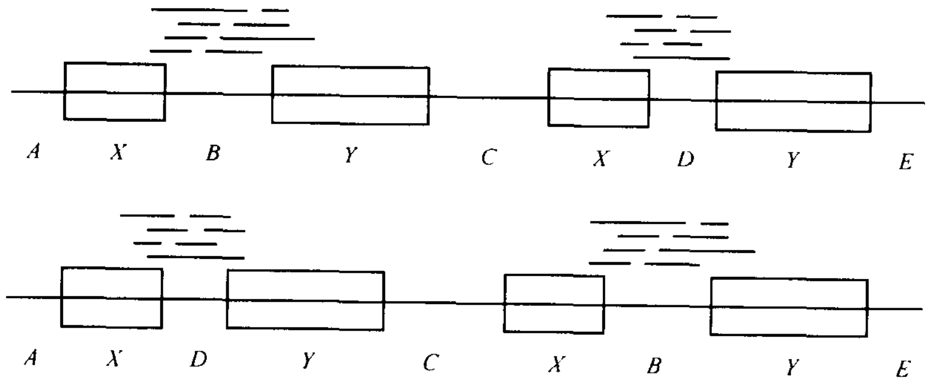


图 4.9　由 $XYXY$ 型的重复，导致的目标序列歧义性组装

到目前为止我们仅讨论了**顺向重复段**(direct repeat)，即重复段仅出现于同一链中。还有一种**逆向重复段**(inverted repeat)，即重复段出现于相反链中，这种重复段更糟糕。
111 只要一个长的逆向重复的两个拷贝就足以产生歧义的目标序列，一个例子见图 4.10。

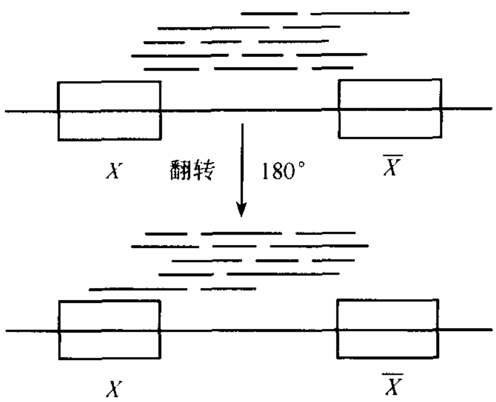


图 4.10　带逆向重复段的目标序列，标 $\overline{X}$ 的区域是标 X 区域的逆的补

= = = = = = = = = = = 缺 乏 覆 盖 = = = = = = = = = = =

组装中的另一个问题是片段间缺乏足够的覆盖。位置 i 的**覆盖**(coverage)指的是在目标序列位置 i 的片段的数量。这个概念的定义是清楚的，但却无法计算，因为我们不知道片段在目标序列中的实际位置。即便组装后，我们也只能有关于这些位置的猜测。但是，我们能够计算平均覆盖，即把所有片段长度累加起来再除以目标序列的大致长度。 112

如果一个或多个位置的覆盖等于 0，则片段集合缺乏足够的信息来组装完全的目标序列。图 4.11 显示了一个例子，其中有未覆盖的区域。在这种情况下，最好的结果是对每一个连续的覆盖区有一个排列，称做**连叠**(contig)。图 4.11 中有两个连叠。

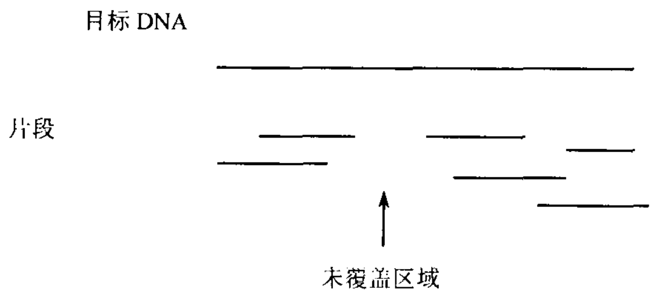


图 4.11　覆盖不足的一个例子

产生覆盖缺乏的原因在于采样。本质上，片段的采样是随机的，因此某些片段的结尾未能被良好地覆盖甚至根本未被覆盖。通常在任一点我们都需要不止一个而是好几个不同片段的覆盖。片段越多，基于多数表决机制的表决构造就越可靠。片段来自两条不同的链覆盖在同一区域也同样必要，因为实践中已观察到某些错误始终只出现在一条链上。

覆盖不足通常可由更多的采样来解决，但对这一措施亦需小心。如果除了目标序列的一小部分外所有区域均覆盖充分，则靠增加随机采样来解决可能是低效的。可以使用另一途径，称做*有向测序*(directed sequencing)或*步移*(walking)(见 4.1.3)。为了避免小空隙的形成，某些研究者提议按照高覆盖率采样，采得的片段总长至少是目标序列的 8 倍以上。

一个重要问题是，我们应该知道需产生多少片段才可达到所需的覆盖。这个问题没

有绝对的答案,因为采样是随机的,但我们可以得出合理的界限。我们这里给出一个结果,它在某些简化的假设下是有效的。

令 T 代表目标分子的长度,且假定所有片段具有相同长度 l 且我们能够肯定地识别至少 t 个碱基的交叠。如果我们随机采样 n 个片段,则*显性连叠*(apparent contig)的期望数 p 是

113 $$p = ne^{-n(l-t)/T} \tag{4.1}$$

术语“显性连叠”指的是这样一个事实,即我们假定我们仅能识别尺度至少不小于 t 的交叠。因此,当某些连叠其交叠小于 t 个碱基时,它们实际上是一个更长的真正的连叠的成分,但却呈现为两个分离的连叠。

注意,随着片段数 n 的增长,p 接近于 0。这看起来令人困惑,因为我们知道至少将有一个显性连叠。之所以产生这个结果,是因为公式(4.1)是一个近似式。该式也能用于 DNA 标图,此时连叠的数量相当可观,因此数量上相差 1 个或 2 个无关紧要。这个公式的有用之处在于它能对广泛的不同情形提供一个可靠的限定。

使用与上面相同的符号,我们有另一个关于目标分子缺隙(fraction)的公式,由恰好 k 个片段进行覆盖的缺隙数是

$$r_k = \frac{e^{-c}c^k}{k!} \tag{4.2}$$

这里 $c = nl/T$ 是平均覆盖。

4.1.3 DNA 测序的其他方法

我们这里介绍 DNA 测序的几个补充方法。首先是有向测序,可用来填补霰弹枪测序的剩余小空隙。在直接测序里,为了产生横跨空隙的片段(包括该连叠的终端及其在目标分子中的接续),应先从连叠的终端导出一个特殊的引物,然后测序这些新片段,得到这个连叠的相邻序列,接着再扩展这个序列。不断重复这个过程,我们能够覆盖当前连叠与下个连叠之间的空隙。使用这个方法存在的问题是构造特殊引物价格昂贵,而且仅在当前步骤完成后才能进行下一步,因此测序过程是顺序而非并行的(但多个空隙的测序可并行进行)。

另一个已非常流行的技术称*双端测序*(dual end sequencing)。在霰弹枪实验里,目标 DNA 分子的几个拷贝随机打散,短碎片用于克隆和测序。我们在 1.5.2 中介绍了这些碎片,它们称做插入片,因为它们被插入载体用于扩增。插入片的大小在 1~5kb 之间,但从一端仅可阅读 200~700 个碱基产生一个片段。然而,如果我们有合适的引物,从另一端阅读也是可行的。所获得的两个片段属于相反链,它们在最后的比对中应按插入片大小减去片段大小而被分开。这种额外信息在填补空隙时格外有用。有时仅当要填补空隙时才使用双端测序。注意双端测序利用了这样一个事实,即插入片通常大于读出部分。

一个完全不同于霰弹枪测序的方法最近被提了出来,称做*杂交测序法*(sequencing by
114 hybridization,SBH),即根据大量短且定长探针的杂交实验组装目标分子。一个杂交仅简单地检测某个探针是否结合(互补)于一个 DNA 分子。整个概念是设计一个 DNA 芯片

来并行完成所有的杂交，并产生一个存在于目标分子中的所有长为 w 的串（w-mer）的表。使用现有技术可以构造探针长度为 8 个碱基的芯片，更大的探针目前尚做不到。

SBH 测序有以下几个问题。首先，很显然并不是每个目标分子都能够用 8 个碱基的小探针重构（见习题 10）。其次，是确定能够用探针重构的分子的特性。再者，杂交实验不能提供 w-mer 在目标分子中的出现次数，而只表示它是否出现。显然，拷贝出现的次数这类额外信息是有用的。最后，必须克服实验过程和方向确定过程中的差错。一个措施是把霰弹枪数据简化为杂交数据，直接使用片段的 w-mer 而不是片段本身，并把它们看做杂交的产物。但这个措施丢弃了有用信息，因为一般我们不可能由 w-mer 复原片段，但可由片段产生 w-mer。解决 SBH 问题的算法未在本书提及，但本章末的文献提要给出了索引。

4.2　模型

我们现在考察片段组装的形式化。这一节有三个模型：*最短公共超串*（shortest common superstring，SCS）、*重构*（reconstruction）、*多连叠*（multicontig），每一个都涉及片段组装的计算方面。所有三个模型均假定片段不含有污染和嵌合片。

4.2.1　最短公共超串

形式化片段组装的最初尝试是通过一个串问题进行的，即在一个给定串集合中寻找最短超串。相应地，它称做最短公共超串问题，或 SCS。尽管这个模型在表达片段组装问题时有严重缺陷，例如它不能处理错误，但所用到的技术可被用于其他模型，因此值得我们进行研究。最短公共超串问题定义如下：

115

问题：最短超串问题（SCS）

输入：串集合 $\mathcal{F}$

输出：一个最短可能超串 S 使得对每一个 $f \in \mathcal{F}$，S 是 f 的一个超串。

例 4.1　令 $\mathcal{F}$ = {ACT, CTA, AGT}，序列 S = ACTAGT 是 $\mathcal{F}$ 的最短公共超串。很明显，它包含 $\mathcal{F}$ 中的所有片段作为子串。为了证实它是最短的，注意到以 u = ACT 和 v = AGT 作为子串的任何串 S' 的长度至少为 6。进一步，对于 $|S'| = 6$，我们有拼接 uv 和 vu。但 CTA 仅是 $uv = S$ 的子串。

在片段组装问题里，集合 $\mathcal{F}$ 对应于片段集，每个片段已知序列和朝向，S 是目标 DNA 分子。

注意该计算问题要求 S 应是每个片段的完全超串，而不是近似超串，因此片段中不应含有错误。进一步，每个片段的朝向必须已知，尽管这常常很难做到。最后，即使所有这些因素全部得到满足，最短公共超串也未必是实际的生物分子，因为 DNA 序列中可能还有重复区域，如下例所示。

例 4.2　假定目标分子中有一个重复序列的两个拷贝，且采样如图 4.12。注意重复序列

拷贝很长,包含了许多片段。在这个例子中,即使片段都是表决序列的真子串,且即使我们知道它们的正确朝向,所得到的最短公共超串也未必是我们所要的。

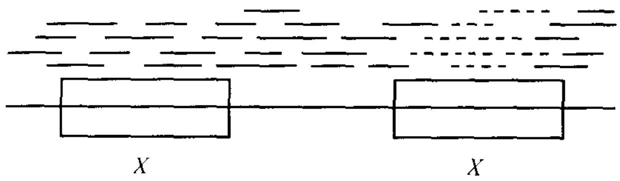


图 4.12　带长重复序列的目标序列。重复序列包含多个片段。这个例子显示即使朝向已知且序列不含错误,重复序列也可导致最短公共超串错误

确实,图 4.13 给出了另一个组装,对同一个片段集合得到了一个更短的表决序列。由于重复序列的拷贝是相同的,超串可以只含有一个拷贝,这个拷贝吸收了所有包含于其他拷贝内部的片段,因而其他拷贝可以更短,因为它(们)仅需包含跨越 X 边界的片段。
116 在本图中缩短的 X 由 X' 表示。

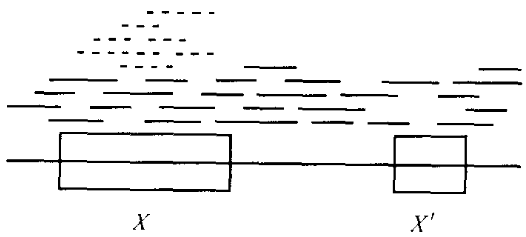


图 4.13　上图中片段的另一种组装。这个组装导致一个更短的表决序列,因为所有包含在重复序列右拷贝内部的片段(虚线片段)都移到了左拷贝,导致右拷贝长度减小

注意,这个候选组装不仅更短,就覆盖与连锁而言,它也更差。覆盖不均衡,多数片段在 X 侧,少量在 X' 侧;连锁也弱,因为没有片段连接 X' 的最左端到最右端。事实上,这个表决序列是两个不相连部分 u 和 v 的拼接,u 从起始直至 X' 的中部,v 则从 X' 的中部到终端。就最短超串而言,vu 也是一个等价表决序列。

最短公共超串问题是 NP-难题,但存在近似算法。然而,面对实际生物学问题,鉴于这个模型所存在的缺陷,算法基本上仅有理论意义。

4.2.2　重构

这个模型考虑了错误和未知朝向。为了处理错误,我们需要几个基本定义。回忆第三章,我们能够对基本的动态程序设计序列比较算法进行配置以适应许多不同的需求。这里,我们用距离取代相似性,并使用一个特殊版本的罚分函数,用以减罚第二个序列末端的空隙。计分系统采用编辑距离,即对每一个插入、删除、替换罚一个距离单元(第二个序列末端的删除是例外),所获得的距离称做*子串编辑距离*(substring edit distance),以区别经典的减罚两串端点删除的编辑距离。我们定义距离 d_s 为

$$d_s(a,b) = \min_{s \in S(b)} d(a,s)$$

这里 $S(b)$ 表示 b 的所有子串的集合,d 是经典的编辑距离。注意子串编辑距离是非对

称的，即一般上 $d_s(a,b) \neq d_s(b,a)$。 117

```
-----GC-GATAG----
CAGTCGCTGATCGTACG
```

图 4.14　用于子串编辑距离的优化比对，对第一个串的终端删除不罚分

例 4.3　如果 a = GCGATAG, b = CAGTCGCTGATCGTACG，则最佳比对如图 4.14 所示，且距离 $d_s(a,b)=2$。

令 ε 是 0 与 1 之间的一个实数，串 f 是 S 的一个具有错误 ε 的近似子串，当

$$d_s(f,S) \leqslant \varepsilon \mid f \mid$$

这里 $|f|$ 是 f 的长度。这意味着对 f 中的每个碱基我们允许平均 ε 的错误。例如，如果 $\varepsilon=0.05$，则我们允许每一百个碱基存在 5 个错误。现在我们定义片段组装的重构模型。

> 问题：重构
>
> 输入：串集合 $\mathcal{F}$ 以及取值 0 与 1 之间的错误容忍度 ε
>
> 输出：一个最短可能串 S 使得对每个 $f \in \mathcal{F}$ 我们有
>
> $$\min(d_s(f,S), d_s(\bar{f},S)) \leqslant \varepsilon \mid f \mid$$
>
> 这里 $\bar{f}$ 是 f 的逆的补。

该算法的思想是找出一个尽可能短的串 S，使得 f 或者 f 的逆补是 S 的一个错误率不超过 ε 的近似子串。这个公式能正确处理 4.1 节中所有的例子，除了包含嵌合片段的例子外。不过，该问题的一般情形仍是 NP-难题性的。这并不奇怪，因为 SCS 正是一个 $\varepsilon=0$ 的特例（见习题 18）。

总之，重构模型可处理错误和朝向，但不能处理重复序列、覆盖缺乏以及目标分子大小。

4.2.3　多连叠

多连叠模型增加了连锁的概念。前面的模型不考虑排列中片段间的关联，仅在意最终的答案。然而，由几个连叠构成的答案也应是可接受的，这正是“多连叠”的起因。

我们首先定义多连叠模型的无错版本。给定一个片段集合 $\mathcal{F}$，考虑一个多序列比对或排列。这个排列必须使得每一列仅包含一种碱基，这便是无错假定的由来。对于朝向问题，我们要求比对包含片段或者其逆补，但不能两者均包含。

称排列为 $\mathcal{L}$，其列数从 1 到 $|\mathcal{L}|$。根据列计数，每个片段 f 有一个左端点 $l(f)$ 和右 118
端点 $r(f)$，因此 $|f| = r(f) - l(f) + 1$。如果整数区间 $[l(f) \ldots r(f)]$ 和 $[l(g) \ldots r(g)]$ 相交，我们就称片段 f 和 g 在这个排列中交叠。非空集合 $[l(f) \ldots r(f)] \cap [l(g) \ldots r(g)]$ 称做 f 与 g 之间的交叠，交叠的大小恰好是交集的大小。因为我们使用整数区间，所有集合都是有限的。

在片段间的所有交叠中，我们对最重要的交叠感兴趣，即提供连锁的交叠。我们说一个交叠 $[x \cdots y]$ 是非连锁的，如果 $\mathcal{F}$ 中存在一个包含交叠两侧的片段，即该片段包含区间

$[(x-1)\dots(y+1)]$。如果$\mathcal{F}$中不存在具有这种性质的片段,则该交叠是一个连锁。排列中的最弱连锁(weakest link)指的是所有连锁中的最小者。最后,我们说一个排列是一个t-连叠(t-contig),如果它的最弱连锁至少与t一样大。如果能用集合$\mathcal{F}$中的片段构造一个t-连叠,则说$\mathcal{F}$容许一个t-连叠。

现在,我们根据多连叠模型来形式化片段组装问题。给定一个片段集合$\mathcal{F}$以及一个整数t,我们要把$\mathcal{F}$划分为数量最少的子集C_i,$1\leqslant i\leqslant k$,使得每个C_i都容许一个t-连叠。

例 4.4 令$\mathcal{F}$ = {GTAC, TAATG, TGTAA},我们要把$\mathcal{F}$划分为最小数量的t-连叠。如果$t=3$,则有如下两个连叠

```
--TAATG      GTAC
TGTAA--
```

无法产生一个包含所有 3 个片段的单个连叠,因此本例中两个连叠是最小数量。注意 GTAC 是它自身的逆补。

如果$t=2$,则上述解也是一个符合条件的解,因为每个 3-连叠同时也是 2-连叠。而另一个解则是

```
TAATG---     GTAC
---TGTAA
```

我们再一次无法把$\mathcal{F}$装配成一个t-连叠。

最后,如果$t=1$,我们看到仅带一个t-连叠的解是

```
TGTAA-----
--TAATG---
------GTAC
```

对容错版本可如下定义。排列不再必须是无错的,但对每一个比对我们有一个同样长度的相关的表决序列S,它可能带有空格。数字$l(f)$和$r(f)$仍然定义为f的最左与最右列,但可能有$|f|\neq r(f)-l(f)+1$。片段f在表决序列中的"映象"序列是$S[l(f)\cdots r(f)]$。给定一个错误容忍度ε,我们说S是一个该连叠下的ε-表决序列,每个序列f与它的在表决序列中的映象之间的编辑距离至多是$\varepsilon|f|$。我们现在给出形式的定义。 119

问题:多连叠

输入:串集合$\mathcal{F}$,整数$t\geqslant 0$,介于 0 与 1 之间的错误容忍度ε。

输出:$\mathcal{F}$的一个最小数量子集的划分C_i,$1\leqslant i\leqslant k$,每个C_i容许一个带ε-表决序列的t-连叠。

注意在重构模型里,我们要求片段是S的近似子串,但其在S中的确切位置则并不重要。相反,在多连叠模型里,我们则需确切地规定片段的起止。

片段组装的多连叠模型是 NP-难题,即使是不含错误和朝向已知的最简单情形亦如此,在一个有限类的图中寻找哈密顿通路是一种特殊情形。

这个模型能处理错误、朝向以及缺乏覆盖,但不能利用目标分子大概长度的信息。此外,它也能一定程度地处理重复序列,因为它能令人满意地解决某些(尽管不是全部)带重

复序列的情形。例如,它能正确地重构图 4.12 中的例子。

*4.3　算法

在这一节我们介绍两个算法,片段是无错的并且朝向是已知的。一个称为贪婪算法(greedy algorithm),许多实际系统均基于相同的思想,但附有处理错误与未知朝向的方法。另一个算法基于多连叠模型,当我们能通过丢弃权小的边获得一个无环交叠图时它非常有用。应强调的是,由于假设的限制,这两个算法原型本身并不具实用价值。然而,其所包含的思想可用于设计能处理实际问题的算法。

4.3.1　交叠的表示

在某些基于$\mathcal{F}$集合的图中,公共超串对应于(图的)通路,我们能把这些超串的属性转换成通路的属性。因为研究者对图的通路研究了很长时间,许多人更愿意处理图,因此把新问题与之相联系常常是好办法。

集合$\mathcal{F}$的**交叠多重图**(overlap multigraph)$\mathcal{OM}(\mathcal{F})$是一个有向带权多重图,定义如下。结点集合 V 就是$\mathcal{F}$本身。对 $a\in\mathcal{F}$, $b\in\mathcal{F}$,如果 a 的 t 字符后缀是 b 的前缀,则 a 120
与 b 间存在一条权 $t\geqslant 0$ 的边,即

$$suffix(a,t)=prefix(b,t)$$

或

$$\kappa^{|a|-t}a = b\kappa^{|b|-t}$$

或

$$(a\kappa^t)b = a(\kappa^t b)$$

κ 是 2.1 节中定义的杀手因子。注意,要使该边存在,我们须有 $|a|\geqslant t$ 和 $|b|\geqslant t$。另外,a 与 b 之间可能存在多条边,它们的 t 值可以不同。这就是为什么这个数据结构称做多重图而不是简单图的原因。注意,自反环(self-loop)是不允许的,即从一个结点出发又回到该结点的边。另一方面,权值为 0 的边是允许的。

关于边的权值要强调的一点是,这些权值代表片段间的交叠,且我们可利用这些交叠来把片段汇合成更长的串。这种汇合意味着我们认为两个片段来自目标 DNA 的相同区域。现在我们必须给出确切的证据。在本例里,证据就是交叠的大小。一个短的交叠提供一个弱的证据,而一个长的交叠则提供一个强的证据。当处理几百个片段,每个片段长几百字符时,认为两个片段间仅共享数个(例如 3 个)字符是不实际的。因此,有时我们要求规定一个最小可允许交叠,而不考虑所有权值小于这个值的边。但原则上我们保留所有的边在多重图中,包括零权值的边。总共有 $n(n-1)$条边,因为任意两个串可享有权为零的边。

4.3.2　通路产生超串

交叠多重图的重要之处在于其所包含的有向通路可产生一个多序列比对。如果存在一个公共超串的话,由该多序列比对又可得到一个表决序列。下面我们更详细地考察这一过程。

给定$\mathcal{OM}(\mathcal{F})$中的一个有向通路 P,我们能依如下方法用 P 中的序列构造一个多序列比对。通路中的每条边 $e=(f,g)$有一个确切的权 t,即边尾 f 的最后 t 个字符是边头 g 的前缀,由此 f 和 g 可对齐,g 开始于 f 末端前的 t 个字符。

图 4.15 是一个例子,图中仅画出了正权值边。集合$\mathcal{F}$是$\{a,b,c,d\}$

$a=$ T A C G A

$b=$ A C C C

$c=$ C T A A A G

121 $d=$ G A C A

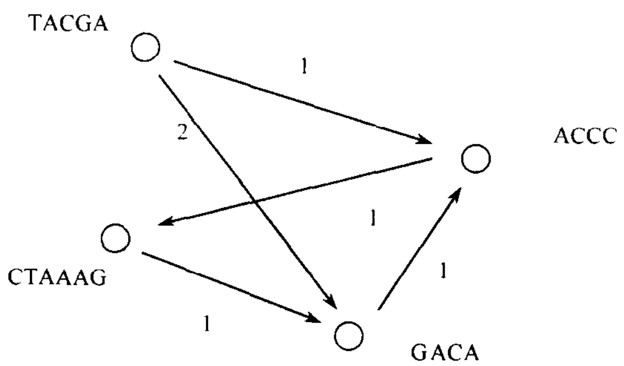


图 4.15 忽略零权值边的交叠多重图

通常我们用一系列的顶点,边,顶点,边……,顶点表示通路,每条边连接两个相邻顶点。然而,在这个例子里没有并行的边,因此为简化起见我们将用一系列顶点表示通路。通路 $P_1=dbc$ 导致如下的比对

```
G A C A - - - - - - - -
- - - A C C C - - - - -
- - - - - - C T A A A G
```

而通路 $P_2=abcd$ 产生如下的比对

```
T A C G A - - - - - - - - - - -
- - - - A C C C - - - - - - - -
- - - - - - - C T A A A G - - -
- - - - - - - - - - - - G A C A
```

通常,令 P 表示$\mathcal{OM}(\mathcal{F})$中的通路,$A$ 表示 P 中所包含的片段集合。通路不能包含重复顶点,因此 P 中恰有$|A|-1$ 条边。由 P 导出的公共超串称为$S(P)$。

A 的全长、通路的权以及超串的长之间的关系是:

$$\|A\| = w(P) + |S(P)| \tag{4.3}$$

这里$\|A\| = \sum_{a\in A}|a|$是 A 中所有序列长度的和,$w(P)$是 P 的权值。下面我们通过例子来检验这个公式。

从仅含单片段 f 且不含边的通路开始,这时 $A=\{f\}$,$\|A\|=|f|$,$w(P)=0$,因为 P 不含边,而$S(P)$也就是f。显然公式(4.3)在此时满足。接着考察规模更大的情况。设

$P=f_1e_1f_2e_2\cdots f_l$，且公式对 P 仍成立。加入一个额外边后情况如何？我们这时有 $P'=Pe_{l+1}f_{l+1}$且

$$w(P')=w(P)+w(e_{l+1})=w(P)+k$$

如果 $w(e_{l+1})=k$。 122

另一方面，$S(P')=S(P)suffix(f_{l+1},|f_{l+1}|-k)$，因为 f_{l+1}的前 k 个字符已经在 $S(P)$中，我们仅需拼接剩余的 f_{l+1}的后缀。这样，

$$\begin{aligned}|S(P')|&=|S(P)|+|f_{l+1}|-k\\&=\|A\|-w(P)+|f_{l+1}|-k\\&=\|A'\|-w(P')\end{aligned}$$

公式仍然成立。我们因此归纳证明了公式(4.3)。

另一个证明的方法是使用杀手字符。如果 $P=f_1e_1f_2e_2\cdots f_l$，则

$$S(P)=f_1\kappa^{w(e_1)}f_2\kappa^{w(e_2)}\cdots f_{l-1}\kappa^{w(e_{l-1})}f_l$$

这里，等式右边并无歧义，因为由边的权定义有

$$(f_i\kappa^{w(e_i)})f_{i+1}=f_i(\kappa^{w(e_i)}f_{i+1})$$

长度的计算是

$$\begin{aligned}|S(P)|&=|f_1|-w(e_1)+|f_2|-w(e_2)+\cdots-w(e_{l-1})+|f_l|\\&=\sum_{i=1}^{l}|f_i|-\sum_{i=1}^{l-1}w(e_i)\\&=\|A\|-w(P)\end{aligned}$$

因此，每一个通路产生一个其所包含序列的公共超串。当我们有一个遍历所有顶点的通路时这便十分重要，这样的通路称哈密顿通路(见 2.2 节)。如果我们有一个哈密顿通路，我们就有一个 $A=\mathcal{F}$ 的公共超串，这时公式 4.3 变成

$$|S(P)|=\|\mathcal{F}\|-w(P)\tag{4.4}$$

且由于 $\|\mathcal{F}\|$ 是常数，即独立于特定的哈密顿通路，我们看到最小化 $|S(P)|$ 就等价于最大化 $w(P)$。

4.3.3　作为通路的最短超串

我们已经看到每一条通路对应一个超串，反之也成立吗？下例说明并不总成立。对产生图 4.15 的片段集，超串

GTATACGACCCAAACTAAAGACAGGG

不对应于任何通路，而且容易看出为什么是这样。超串可能包含不在任何片段中出现的不必要字符。

但最短超串是不能有无用字符的。那么，最短超串是否总是与通路对应？答案是肯定的，我们将证明这一点。我们首先观察到，对某些无子串(substring-free)的集合答案是成立的，然后把结论推广到任意集合。 123

一个集合 $\mathcal{F}$ 称做是无子串的，如果不存在两个串 a 和 b 使得 a 是 b 的子串。使用无子串集合的好处是由通路到超串的部分收敛(partial converse)容易被证明。

定理 4.1 令$\mathcal{F}$是一个无子串集合。对$\mathcal{F}$中的每一个公共超串 S,在$\mathcal{OM}(\mathcal{F})$中存在一个哈密顿通路 P 使得$S(P)$是 S 的一个子序列。

证明:串 S 是$\mathcal{F}$的公共超串,因此对每个 $f\in\mathcal{F}$ 存在 S 的一个区间$[l(f)\cdots r(f)]$使得$S[l(f)\cdots r(f)]=f$。某些片段可在 S 中有多个位置,这并不要紧,取一个即可。

因为$\mathcal{F}$是无子串的,没有一个形如$[l(f)\cdots r(f)]$的区间被包含在另一个区间里。这些区间具有如下重要性质:所有的左端点和右端点都是惟一的,尽管有可能左端点等于右端点。进一步来说,如果我们按左端点的升序排序片段,右端点也形成升序。的确,如果我们有$\mathcal{F}$中的两个串 f 和g,且 $l(f)\leqslant l(g)$和 $r(f)\leqslant r(g)$,则 g 的区间将被包含在f 的区间中,而这是不可能的。因此我们有起点与终点均按升序排序的区间。令$(f_i)_{i=1,\cdots,m}$是所排的序,则序中的片段构成相应的通路。

考虑 f_i 与f_{i+1}之间的关系,对每一个 i 有 $1\leqslant i\leqslant m-1$。如果 $l(f_{i+1})\leqslant r(f_i)+1$,则$\mathcal{OM}(\mathcal{F})$中存在一个边其权为 $r(f_i)+1-l(f_{i+1})$。如果 $l(f_{i+1})>r(f_i)+1$,则从 f_i 到f_{i+1}的边的权是 0。满足 $r(f_i)+1\leqslant j\leqslant l(f_{i+1})-1$ 的字符 $S[j]$将被丢弃,因为它们不属于任何区间,因此对于构造公共超串是多余的。这些字符在 S 中但不在$S(P)$中,后者因此是 S 的子序列。 ■

推论 4.1 令$\mathcal{F}$是一个无子串集合。如果 S 是$\mathcal{F}$的一个最短公共超串,则存在一个哈密顿通路使得 $S=S(P)$。

证明:由以上定理知,存在一个 S 的子序列$S(P)$的通路。但 $S(P)$也是一个公共超串,因为 P 是哈密顿通路。因为 S 是一个最短公共超串,我们有$|S(P)|\geqslant|S|$且因此 $S=S(P)$。 ■

现在把这个结果推广到任意集合。首先需要几个定义。当每个 $g\in\mathcal{G}$ 都是某个 $f\in\mathcal{F}$的子串时,称串集合$\mathcal{F}$支配串集合$\mathcal{G}$。例如,如果$\mathcal{G}\subseteq\mathcal{F}$则$\mathcal{F}$支配$\mathcal{G}$。如果$\mathcal{F}$支配$\mathcal{G}$而$\mathcal{G}$也支配$\mathcal{F}$,则两个串集合$\mathcal{F}$和$\mathcal{G}$称为等价的,表示为$\mathcal{F}\equiv\mathcal{G}$。等价集合具有相同的超串。

引理 4.1 两个等价无子串集合是相同的。

证明:如果$\mathcal{F}\equiv\mathcal{G}$且$\mathcal{F}$是无子串的,则$\mathcal{F}\subseteq\mathcal{G}$。为了证明这一点,考虑 $f\in\mathcal{F}$。因为$\mathcal{G}$支
124 配$\mathcal{F}$,必有一个 $g\in\mathcal{G}$ 使得 f 是g 的一个子串。但$\mathcal{F}$也支配$\mathcal{G}$,因此$\mathcal{F}$中也存在一个 g 的超串h。由传递性,f 是h 的一个超串。但因$\mathcal{F}$是无子串的,因此 $f=h$,且因为 g 在它们之间,因而有 $f=g=h$。这导致 $f=g\in\mathcal{G}$。由于 f 是任意的,我们得到$\mathcal{F}\subseteq\mathcal{G}$。交换$\mathcal{F}$和$\mathcal{G}$同样可证明$\mathcal{G}\subseteq\mathcal{F}$,因此$\mathcal{G}=\mathcal{F}$。 ■

下面的定理帮助我们理解如何从任意片段集合获得无子串集合。

定理 4.2 令$\mathcal{F}$是一个串集合,则存在一个惟一的无子串集合$\mathcal{G}$等价于$\mathcal{F}$。

证明:惟一性由先前的引理保证。现在证明存在性,我们需要证明对每一个$\mathcal{F}$存在一个等价于它的无子串集合$\mathcal{G}$。我们通过对$\mathcal{F}$中串的数量进行归纳来证明。

对$|\mathcal{F}|=0$,$\mathcal{F}$本身是无子串的,因此得证。对$|\mathcal{F}|\geqslant 1$,$\mathcal{F}$可以是或不是无子串的。如果它是无子串的,则得证;如果它不是,则存在$\mathcal{F}$中的 a 和 b 使得 a 是 b 的子串。从$\mathcal{F}$中移去 a,得到一个更小的集合$\mathcal{F}'=\mathcal{F}-\{a\}$。由归纳假设,$\mathcal{F}'$有一个等价的无子串集合$\mathcal{G}$。容

易检验$\mathcal{F} \equiv \mathcal{F}'$。明显地,$\mathcal{F}$ 支配$\mathcal{F}'$,因后者是$\mathcal{F}$ 的一个子集。另一方面,惟一地存在于$\mathcal{F}$ 中但不存在于$\mathcal{F}'$中的串是 a,它是 $b \in \mathcal{F}'$的一部分。因此,$\mathcal{G}$ 总是等价于$\mathcal{F}$,由此得证。 ■

这个结果告诉我们,如果要寻找公共超串,我们亦可限于无子串集合,因为对任何集合都存在一个等价于它的无子串集合,有着相同的公共超串。定理 4.2 的证明恰好也给了我们一个获得等价于$\mathcal{F}$ 的惟一的无子串集合的方法:移去$\mathcal{F}$ 中所有是其他串子串的串。

4.3.4 贪婪算法

我们现在知道寻找最短公共超串与在有向多图中寻找权值最大的哈密顿通路是一样的。进一步,由于我们的目标是最大化权值,我们可以简化多图,仅考虑每一对顶点间最重的边而丢弃小的重叠边。凡未历经最重边的通路均有待改进,因为它们不是最大权通路。我们称这一新图为$\mathcal{F}$ 的*交叠图*(overlap graph),表示为 $\mathcal{OG}(\mathcal{F})$。在一个实际的算法实现中我们并不逐一构造这些边。使用后缀树(3.6.3)可以使我们并行地构造这些边,既节省空间也节省时间。同样的结构还可帮助我们确定哪些片段是其他片段的子串,进而将它们排除在图外。

下面的算法是个"贪婪"的算法,计算最重通路。其基本思想是不断加入最重的边,且不破坏已完成部分的哈密顿通路。因为图是完全的,即每一对顶点间均存在边,故当通路 125
囊括进所有顶点时算法便告终止。

在一个哈密顿通路或任何类似通路里,任一顶点不能发出两条边,也不能进入两条边,此外我们必须防止环的形成。这是在构造哈密顿通路时接受一条新的边之前必须判明的三个条件。按权的非增序处理边,直至图中有 $n-1$ 条边,或新接受的边导致一个连通子图。

图 4.16 展示了一个算法。对每个结点我们利用边的信息记下其当前的入度和出度,

```
Algorithm Greedy
    input: weighted directed graph OG(F) with n vertices
    output: Hamiltonian path in OG(F)
    // Initialize
    for i ← 1 to n do
        in[i] ← 0 // how many selected edges enter i
        out[i] ← 0 // how many selected edges exit i
        MakeSet(i)
    // Process
    Sort edges by weight, heaviest first
    for each edge (f, g) in this order do
        // test edge for acceptance
        if in[g] = 0 and out[f] = 0 and FindSet(f) ≠ FindSet(g)
            select (f, g)
            in[g] ← 1
            out[f] ← 1
            Union(FindSet(f), FindSet(g))
        if there is only one component
            break
    return selected edges
```

图 4.16　发现哈密顿通路的贪婪算法

126

这个信息用于检查一个先前选择的边是否与当前检查中的边有相同的尾或头。此外，我们保留由已接受边构成的图的连通成分的不相交集。容易发现，当且仅当一个新边的尾和头在同一个集合中时这个新边导致一个环。因此这个结构对于环的检测是必要的。不相交集由一个称为不相交集合森林的数据结构维持(见 2.3 节)。

我们用图的术语描述贪婪算法，不过它能直接用片段实现。算法的过程可连续地用于片段的处理，直至仅剩一个片段。选出交叠 k 最大的片段对(f,g)，将它们移出$\mathcal{F}$，将 $f\kappa^k g$ 加入$\mathcal{F}$中。我们假定集合$\mathcal{F}$是无子串的。

如下例所示，贪婪算法的策略并不总是产生最佳结果。

例 4.5　假定我们有

$$\mathcal{F} = \{GCC, ATGC, TGCAT\}$$

交叠图如图 4.17 所示，图中我们忽略了权为 0 的边。

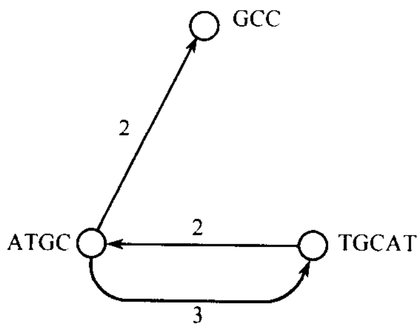


图 4.17　贪婪算法失效的一个图

在这个图中，权为 3 的边是第一个被考察的，且被接受，因为第一个边总是被接受的。然而，它使两条权为 2 的边被废弃，算法不得不选择一条权为 0 的边以完成哈密顿通路。两个被废弃的边构成一个总权为 4 的通路，而它才是这个例子中最重的通路。

这个例子显示贪婪算法并不总是返回最短超串。存在一个在各种情况下均能返回最短超串的算法吗？很明显不存在。我们试图通过哈密顿通路解决 SCS 难题，但如在 2.3 节中所见，哈密顿难题是 NP-完全的。或许我们能够幸运地有其他途径解决 SCS 难题，不幸的是，这也不大可能，因为，如前所述，SCS 难题是 NP-难题。

4.3.5　无环子图

前一部分的算法可用于任意的片段集合。在这一部分我们考虑不带错误和朝向已知的片段的组装问题，这些片段由目标 DNA 的“好”的采样获得。后面我们将看到，这两个假定使我们能够导出既正确又高效的算法。

“好”采样的指标是什么？基本上，我们要求片段覆盖整个目标分子，且集合作为一个整体能有足够的连锁以保证可靠的组装。回想 4.2.3 中 t-连叠的定义，我们这里的定义也基于类似的概念。

127 假设 S 是字母表$\{A,C,G,T\}$上的串。在 2.1 节中，我们说 S 的一个区间实际上是一个整数区间$[i...j]$，$1\leqslant i\leqslant j+1\leqslant |S|+1$。$S$ 的一个采样是 S 的一个区间集合$\mathcal{A}$。如果对任意的 i，$1\leqslant i\leqslant |S|$，我们至少有一个区间$[j...k]\in\mathcal{A}$，$i\in[j...k]$，则说采样$\mathcal{A}$覆盖 S。

如果$|\alpha\cap\beta|\geqslant t$，就称两个整数区间 α 和β 在层次 t 连锁。如果对$\mathcal{A}$中每一对区间 α 和 β，都存在一个区间系列 α_i，$0\leqslant i\leqslant l$，使得 $\alpha=\alpha_0$，$\beta=\alpha_l$，且 α_i 与 α_{i+1}之间有层次 t 的连锁$(0\leqslant i\leqslant l-1)$，则称整个采样$\mathcal{A}$在层次 t 互连。注意我们使用参数 t 度量连锁的强度。某个采样$\mathcal{A}$被认为是“好”采样，如果它在层次 t 互连。

我们的目标是研究采样的片段集合，这个集合在层次 t 互连并覆盖某个串 S。在实际情形中，典型的 t 值在 10 左右，但在这里我们不计较具体的值。当然，我们还假定 t 是一个非负整数。

由采样 $\mathcal{A}$ 产生的片段集合是：

$$S[\mathcal{A}] = \{S[\alpha] \mid \alpha \in \mathcal{A}\}$$

如果在 $\mathcal{A}$ 中不存在两个区间 $[i...j]$ 和 $[k...l]$ 使得 $[i...j] \subseteq [k...l]$，则称采样 $\mathcal{A}$ 是无子区间的(subinterval-free)。下面关于无子区间采样的特性将被反复用到。在一个无子区间采样里，没有两个区间有相同的左端点，否则，右端点较大的那个区间将包含右端点较小的那个区间。同样，亦没有两个区间有相同的右端点。

我们考虑修正的交叠图。给定一个集合 $\mathcal{F}$ 和一个非负整数 t，用 $\mathcal{OM}(\mathcal{F}, t)$ 定义由 $\mathcal{OM}(\mathcal{F})$ 去掉权值小于 t 的边所获得的交叠图。类似地可构造 $\mathcal{OG}(\mathcal{F}, t)$。这样减少了整个边的数量，仅保留了权较重的边。

我们将使用记号 $\alpha \xrightarrow{w} \beta$ 表示 $|\alpha| \geq w$，$|\beta| \geq w$，$l(\beta) + w - 1 = r(\alpha)$。类似地，用记号 $f \xrightarrow{w} g$ 表示 $\mathcal{OM}(\mathcal{F})$ 中存在一个权为 w 的由 f 到 g 的边。

假定我们有了集合 $\mathcal{F}$，要求构造 S，如何进行？什么是好的算法？答案取决于 S 中的重复结构。如果 S 没有任何大小如 t 或更大的重复结构，则构造 S 是简单的。但如果 S 有这种重复结构，问题就复杂了。在某些情形下 S 无法被无歧义地重构。

与 S 中重复结构无关的是图 $\mathcal{OM}(\mathcal{F}, t)$ 中哈密顿通路的存在性。我们在这里给出基本的证明。首先是下面的引理。

引理 4.2　S 是 $\{A, C, G, T\}$ 上的串，$\mathcal{A}$ 是一个无子区间的，在层次 t 互连的 S 的采样，$t > 0$。如果 α 是 $\mathcal{A}$ 中的这样一个区间，它使 $\mathcal{A}$ 中存在另一个区间 β，且 $l(\alpha) < l(\beta)$，则可以选择 β，使得

$$r(\alpha) + 1 - l(\beta) \geq t$$

证明：我们知道 $\mathcal{A}$ 在层次 t 互连，因此在 $\mathcal{A}$ 中存在一个区间序列 $(\alpha_i) 0 \leq i \leq l$，使得 $\alpha = \alpha_0$，$\beta = \alpha_l$，且 $|\alpha_i \cap \alpha_{i+1}| \geq t$ 对对 $0 \leq i \leq l - 1$。

令 i 是满足 $l(\alpha) < l(\alpha_i)$ 的最小下标。因为 $\beta = \alpha_l$ 具有这一性质，故我们知道 $i \leq l$。另一方面，因为 $\alpha = \alpha_0$ 不具有这一性质，故 $i > 0$。因为 i 是具有这种性质的最小下标，我们有 $l(\alpha_{i-1}) < l(\alpha) < l(\alpha_i)$。因为 $\mathcal{A}$ 是无子区间的，它也满足 $r(\alpha_{i-1}) < r(\alpha) < r(\alpha_i)$，如图 4.18 所示。 128

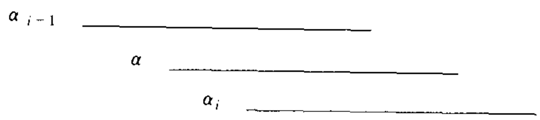


图 4.18　三个相互交叠的区间

我们已知 $|\alpha_{i-1} \cap \alpha_i| \geq t$，但 $|\alpha \cap \alpha_i| \geq |\alpha_{i-1} \cap \alpha_i| \geq t$(见图 4.18)，因此 α_i 是 $\mathcal{A}$ 的一个满足下式的区间

$$r(\alpha) + 1 - l(\alpha_i) \geq t$$

■

现在，证明哈密顿通路的存在性。

定理 4.3　令 S 是 $\{A,C,G,T\}$ 上的一个串，$\mathcal{A}$ 是一个无子区间的在层次 t 互连的 S 的采样，$t\geqslant 0$。则多图 $\mathcal{OM}(\mathcal{F},t)$ 存在一个哈密顿通路，$\mathcal{F}$ 是 $\mathcal{A}$ 产生的片段集合。进一步，如果 $\mathcal{A}$ 覆盖 S，则存在 P 使得 $S(P)=S$。

证明： 令 $(\alpha_i)_{1\leqslant i\leqslant n}$ 是 $\mathcal{A}$ 中所有的区间，且按左端点排序。因为 $\mathcal{A}$ 是无子区间的，这个序列也按右端点排序。令 $f_i=S[\alpha_i]$，$1\leqslant i\leqslant n$。我们要求对某些 $k_i\geqslant t$ 有 $f_i\xrightarrow{k_i}f_{i+1}$，使得 f_i 构成一个哈密顿通路。

令 α_i 是除最后一个区间外的任意区间。由先前的引理，存在另一个区间 $\beta\in\mathcal{A}$ 使得 $l(\alpha_i)<l(\beta)$ 且 $r(\alpha_i)+1-l(\beta)\geqslant t$。序列 $(\alpha_i)_{1\leqslant i\leqslant n}$ 是按左端点排序的，因此 $l(\alpha_i)<l(\alpha_{i+1})\leqslant l(\beta)$（区间 α_{i+1} 和 β 可以是相同的）。因此

$$\begin{aligned} k_i &= |\alpha_i\cap\alpha_{i+1}| = |[l(\alpha_{i+1})\cdots r(\alpha_i)]| \\ &= r(\alpha_i)-l(\alpha_{i+1})+1\geqslant r(\alpha_i)+1-l(\beta)\geqslant t \end{aligned}$$

这显示一个边 $f_i\to f_{i+1}$ 确实在 $\mathcal{OM}(\mathcal{F},t)$ 中存在，定理的第一部分得证。

定理的第二部分，即 $S(P)=S$，是下面事实的自然结果。每当我们有 $\alpha\xrightarrow{w}\beta$，我们便有 $S[\alpha]\xrightarrow{w}S[\beta]$ 和 $S[\alpha]\kappa^w S[\beta]=S[\alpha\cup\beta]$。　■

目标串中重复区域的出现与交叠图中环的存在相关。这一部分以及下一部分我们着重讨论这种关系。

当 S 中的两个不同区间产生相同子串时，S 中就出现一个重复片段。换言之，存在两个区间 $\alpha\neq\beta$ 使得 $S[\alpha]=S[\beta]$。重复区间的大小是 $|S[\alpha]|$，它和 $|S[\beta]|$ 一样大。当
129 $\alpha\cap\beta\neq\phi$ 时，这个重复片段是自交叠(self-overlapping)的，否则，它们分别出现。

我们的下一个结果显示，交叠图中的环必然是由于 S 中的重复片段引起的，但重复片段的收敛则并不必然出现，即我们可以有重复序列，但交叠图仍然无环。在我们证明这个重要结果之前，我们仍需若干中间结果。

引理 4.3　如果 $\alpha\xrightarrow{w}\beta$ 则 $l(\alpha)\leqslant l(\beta)$ 并且 $r(\alpha)\leqslant r(\beta)$。

证明： 从 $|\alpha|\geqslant w$ 我们有 $l(\alpha)+w-1\leqslant r(\alpha)$。从 $\alpha\xrightarrow{w}\beta$ 我们有 $l(\beta)+w-1=r(\alpha)$。因此 $l(\alpha)+w-1\leqslant l(\beta)+w-1$，即 $l(\alpha)\leqslant l(\beta)$。类似地，$|\beta|\geqslant w$ 意味着 $l(\beta)+w-1\leqslant r(\beta)$，或 $r(\alpha)\leqslant r(\beta)$，考虑到先前的方程。　■

接着，我们需要几个定义。在层次 t 的假阳性(false positive)指的是有一对区间 α 和 β，且有一个 $w\geqslant t$ 使得 $S[\alpha]\xrightarrow{w}S[\beta]$，但却没有 $\alpha\xrightarrow{w}\beta$。这个名称反映了一个事实，即由于它们的序列交叠，我们假定 $S[\alpha]$ 和 $S[\beta]$ 来自于目标分子中的交叠区域，而实际并非如此。它们看起来是交叠的但实际不是，故称为假阳性。

引理 4.4　假阳性在层次 t 的存在意味着一个规模至少为 t 的重复片段的存在。

证明： 因为 $S[\alpha]\xrightarrow{w}S[\beta]$，我们有 $|\alpha|\geqslant w$ 和 $|\beta|\geqslant w$。如果 $l(\beta)=r(\alpha)-w+1$，因为假定 $\alpha\neq\beta$，则我们将有 $\alpha\xrightarrow{w}\beta$。因此 $l(\beta)\neq r(\alpha)-w+1$。

关系 $S[\alpha]\xrightarrow{w}S[\beta]$ 也意味着 $S[\alpha']=S[\beta']$，这里 α' 是 α 的最后 w 个字符，β' 则是 β

的开头 w 个字符。因为 $l(\beta)\neq r(\alpha)-w+1$,所以 α' 和 β' 是产生相同子串的不同区间,即我们有一个规模为 $w\geq t$ 的重复片段。 ■

定理 4.4　令 $\mathcal{F}$ 是 S 的采样 $\mathcal{A}$ 产生的集合。如果 $\mathcal{OG}(\mathcal{F},t)$ 有一个有向环,则 S 中存在一个规模至少为 t 的重复片段。

证明:令 $f_1\rightarrow f_2\rightarrow\cdots f_l\rightarrow f_1$ 是 $\mathcal{OG}(\mathcal{F},t)$ 的一个有向环。考虑区间 $\alpha_1\cdots\alpha_l$ 使得 $S[\alpha_i]=f_i$,对于 $1\leq i\leq l$。我们将证明 (α_i,α_{i+1})[包括了 (α_l,α_1)]中的至少一对是一个假阳性。由引理 4.4,这意味着 S 中一个重复片段的存在。

假定不存在假阳性,则对 $1\leq i<l$ 存在着 $\alpha_i\rightarrow\alpha_{i+1}$ 以及 $\alpha_l\rightarrow\alpha_1$,因而有 $l(\alpha_1)\leq l(\alpha_2)\leq\cdots\leq l(\alpha_l)\leq l(\alpha_1)$。因此,所有的 $l(\alpha_i)$ 都是均等的。类似的观察告诉我们对 $r(\alpha_i)$ 也成立。接着便有 $\alpha_1=\alpha_2=\cdots=\alpha_l$,但这是不可能的,因为按照假设 f_i 是互不相同的。因此,存在至少一个假阳性,结果得证。 ■

当目标串 S 不存在重复片段时,这些论证导致片段组装的一个满意解。

定理 4.5　S 是一个 $\{A,C,G,T\}$ 上的串,$\mathcal{A}$ 是一个覆盖 S 的采样,无子区间且在层次 t 互连,$\mathcal{F}=S[\mathcal{A}]$。如果 S 没有规模为 t 或大于 t 的重复片段,则图 $\mathcal{OG}(\mathcal{F},t)$ 有一个惟一的哈密顿通路 P,且 $S(P)=S$。 130

证明:我们已知 $\mathcal{OG}(\mathcal{F},t)$ 是无环的(定理 4.4),且 $\mathcal{OM}(\mathcal{F},t)$ 有一个哈密顿通路 P 满足 $S(P)=S$。

首先注意到,如果 S 没有规模为 t 或大于 t 的重复片段,则 $\mathcal{OM}(\mathcal{F},t)=\mathcal{OG}(\mathcal{F},t)$。由于没有假阳性,$\mathcal{OM}(\mathcal{F},t)$ 的顶点集和 $\mathcal{OG}(\mathcal{F},t)$ 的顶点集显然是相同的。同时,$\mathcal{OG}(\mathcal{F},t)$ 的每一条边也在 $\mathcal{OM}(\mathcal{F},t)$ 中。我们只需确证 $\mathcal{OM}(\mathcal{F},t)$ 中的每一条边也同样在 $\mathcal{OG}(\mathcal{F},t)$ 中。

令 $f\xrightarrow{w}g$ 是 $\mathcal{OM}(\mathcal{F},t)$ 中的一条边,这样便有 $w\geq t$。如果这条边不在 $\mathcal{OG}(\mathcal{F},t)$ 中,则有另一条边 $f\xrightarrow{x}g$ 存在,$x>w$。令 α 和 β 是 $\mathcal{A}$ 中的区间,$S[\alpha]=f$ 并且 $S[\beta]=g$。由于不存在假阳性,我们便同时有 $\alpha\xrightarrow{w}\beta$ 和 $\alpha\xrightarrow{x}\beta$,而这是不可能的。这条边的权是惟一地由 $w=r(\alpha)-l(\beta)+1$ 确定的。

因此,$\mathcal{OG}(\mathcal{F},t)$ 是一个带哈密顿通路的无环图,它表明每当一个无环图有一个哈密顿通路时这个通路是惟一的。为了说明为何如此,考虑图中入度为零的结点(源 source)。因为图是无环的,它必须至少有一个源。但是,因包含一个哈密顿通路,它又只能至多有一个源,即通路的起点。所有其他的顶点则至少有一个入边。结论是我们的图必须恰有一个源,且它必是哈密顿通路的起点。

现在如果我们从图中移去这个源,剩余的图仍然是无环的,且有一个哈密顿通路,因为移去的顶点是原有哈密顿通路的顶端。继续这个过程,我们看到哈密顿通路是惟一的,且我们知道如何构造它:不断地从图中移出源。 ■

以上描述的算法即是计算机科学中有名的拓扑排序法(见第 2 章习题 12)。在更一般的情形里,它被用来发现一个与无环边集合一致的结点排序,且对所有的边 $f\rightarrow g$,结点 f 位于结点 g 之前。如果除了无环图外还有一个哈密顿通路,则这个排序就是惟一的。

我们用一个例子结束这一部分。贪婪算法对这个例子并不适用,但对某个合适的阈

值 t 存在一个无环子图，使得拓扑排序能够发现正确的通路。当我们说贪婪算法不适用时，是指它无法用于片段组装；而对于发现最短公共超串它总是胜任的。在这个例子里，产生片段的目标序列确实包含长于 t 的重复片段。长片段的缺失并不必然产生无环图。用数学术语说，这仅是充分条件。

例 4.6 考虑如下的目标序列 S 及片段 w, z, u, x, y：

S = AGTATTGGCAATCGATGCAAACCTTTTGGCAATCACT

w = AGTATTGGCAATC

z = AATCGATG

u = ATGCAAACCT

x = CCTTTTGG

131 y = TTGGCAATCACT

选择 t 的阈值 $t=3$，我们得到图 4.19 所示的交叠图。一个贪婪组装算法将导致一个最短公共超串，由连叠 $w\kappa^9 y$ 与连叠 $z\kappa^3 u\kappa^3 x$ 组成，见图 4.20 和图 4.21。这个解法得到的目标串长 36，短于目标串 1 个字符(也是一个超串)。而由哈密顿通路给出的解有更好的关联。

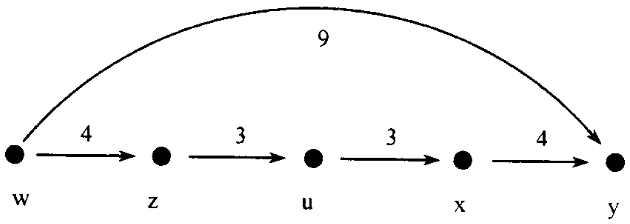


图 4.19 给定 5 片段的交叠图

```
AGTATTGGCAATC---AATCGATG------------
---------------------ATGCAAACCT-----
----TTGGCAATCACT------------CCTTTTGG
AGTATTGGCAATCACTAATCGATGCAAACCTTTTGG
```

图 4.20 片段组装问题的一个坏解。多序列比对的表决序列是一个最短公共超串。这个解长 36，由贪婪算法产生，然而它的最弱连接是 0

```
AGTATTGGCAATC--------CCTTTTGG--------
---------AATCGATG--------TTGGCAATCACT
--------------ATGCAAACCT-------------
AGTATTGGCAATCGATGCAAACCTTTTGGCAATCACT
```

图 4.21 根据惟一哈密顿通路产生的解，长度 37，

132 但有更好的关联。它的最弱连接(weakest link)是 3

4.4 启发式

我们已经看到没有一个形式化的方法对片段组装是完全足够的，因此，实际上我们必

须经常借助于启发式方法。我们在这一部分评估某些启发式方法,并指出一个好的片段排列所具有的性质。

片段组装可看做具有某些附加特性的多序列比对。最突出的特性也许是这个事实,即每个片段既可以直接序列加入亦可以逆补序列加入。第二个特性是序列本身通常大大短于比对分子,这导致我们对片段的内部空隙和外部空隙进行不同的罚分,内部空隙罚分更高。如果我们对两种空隙的罚分相同或接近,则使片段字符集中相连的因素便大大削弱,从而导致它们散布在比对的整个长度上。更进一步,片段大大短于比对这一事实促使我们寻找计分以外的其他标准来评估比对的质量,这些额外的标准包括覆盖和连锁(linkage),后者用片段交叠的强度来度量序列间的相关性。我们现在具体地讨论这些标准。

计分(scoring):在多序列比对的每一列中我们都有某种程度的覆盖(4.1.2 中定义)。理想的情形是,构成一列的来自所有序列的字符是相同的。一般地,一致的列好于有差异的列。

这样导致使用一个称做熵(entropy)的概念来度量列的质量。熵是一个定义在一组相对频率上的量。当某个频率突出于其他频率时,熵是低的;当所有频率均相差无几时,熵值则高。在我们这里,熵越低越好。

假定我们有 n 个序列覆盖某一列,每一个字符的相对频率是 p_A, p_C, p_G, p_T 以及 p_{space},这个分布的熵是

$$E = -\sum_c p_c \log p_c$$

如果对某一字符 c 有 $p_c = 1$ 且所有其他频率为 0,则我们有(假定 $0\log 0 = 0$)

$$E = 0$$

另一方面,如果所有的频率都相等,我们对每个 c 都有 $p_c = 1/5$,且

$$E = -5 \times \frac{1}{5}\log\frac{1}{5} = \log 5$$

这是对 5 个频率的熵的最大可能值。 133

回忆 3.4 节,当讨论多序列比对时,我们使用配对和计分机制(sum-of-pair),这个办法用一个数整合了覆盖和列的一致性。如果一致性高,则覆盖越大,配对和值也越大。而且,对于恒常覆盖(constant coverage),配对和值随一致性的增加而增加。与之相反,熵则独立于覆盖,它仅度量一致性。鉴于此,科学家可分别使用组合性的度量配对和,或覆盖加上熵,来进行计分。

覆盖(coverage):在 4.1.2 我们定义了一个片段集的覆盖。这里,对排列我们定义一个类似的概念。一个片段覆盖一列 i,如果它或以字符或以内部空格参与该列。令 $l(f)$ 和 $r(f)$ 是片段 f(或它的逆补)的字符在排列中的最左位置和最右位置。上面的定义意味着片段 f 覆盖列 i,如果 $l(f) \leqslant i \leqslant r(f)$。

定义了每一列的覆盖后,我们就可以讨论一个排列的最小、最大和平均覆盖。如果某列的覆盖值为 0,则我们并没有一个真正互连的排列。的确,没有片段跨越列 i 意味着排列的列 i 的左边和右边是独立的。例如,右边部分可被换到左边,产生一个与原排列一样好的候选排列。

这意味着如果我们有更多的零覆盖列,则相间区域的任何置换都是可接受的。每个

这样的区域对应于一个连叠。覆盖对表决也是重要的，如果有几个片段都覆盖一个列，则我们对该列的碱基就有更多的信息。有足够的覆盖片段来自排列的两端也很重要。

```
------ACTTTT------
TCCGAG------ACGGAC
------ACTTTT------
TCCGAG------ACGGAC
------ACTTTT------
TCCGAG------ACGGAC
------------------
TCCGAGACTTTTACGGAG
```
134

图 4.22　好的覆盖但差的连锁

连锁(linkage)：除了计分和覆盖，片段在排列中的连接方式是另一个决定排列质量的因素。例如，考虑图 4.22 的排列，覆盖是高的，但片段间没有真正的关联，每一块都由几乎一样的拷贝组成，而块之间几乎没有联系。片段间应有交叠段，以显示连锁的证据。

一个好的组装程序必须平衡这些需求，且找到能合理地满足它们的所有解。这不是一件容易的事，况且不存在完美的形式解，因此算法的设计尤其困难。其结果是，在实际实现时经常将问题分为 3 部分，每一部分使用前一阶段的结果。一个典型的划分包括下列阶段：

- 发现交叠
- 构造排列
- 计算表决序列

这种划分使得问题更易于处理，因为每个阶段能够当做不同问题独立处理。然而，理解初始输入和最终输出之间的关系却更困难了。用于这些阶段的方法常常是启发式的，对解的质量无法有清晰的保证。不过，许多现有的算法在实际中运行的相当好，因而值得了解其中的一些技术。

4.4.1　发现交叠

任何组装程序的第一步都是片段交叠检测。我们需要检查所有的片段以及它们的逆补以确定是否某个片段的前缀与另一个片段的后缀匹配；同时，我们也注意一个片段完全为其他片段包含的情形。所有这些比较都控制在某一个错误阈值范围内。

这里介绍一个比较两个片段并发现交叠的方法，在实际问题里我们可将这个方法应用到所有片段对，包括逆补，但实际上我们很少这么做，因为平方复杂性的算法耗时太多。不过，它是最可靠的交叠评估方法之一。

比较方法包括 3.2.3 中介绍的动态程序设计算法，它是基本算法的一个变体，这个变
135 体对两端的空格不罚分。图 4.23 给出了使用这种方法的结果，计分系统是 1 表示匹配，-1 表示失配，-2 表示空格。注意算法对第一个序列的后部以及第二个序列的前部的空格没有罚分，因此它可以发现前缀 - 后缀相似性。同样的算法也可以发现包含关系。

```
AGGAGAAGAATTCACCGCTAT----------
----------TTCCCCT-TATTCAATTCTAA
```

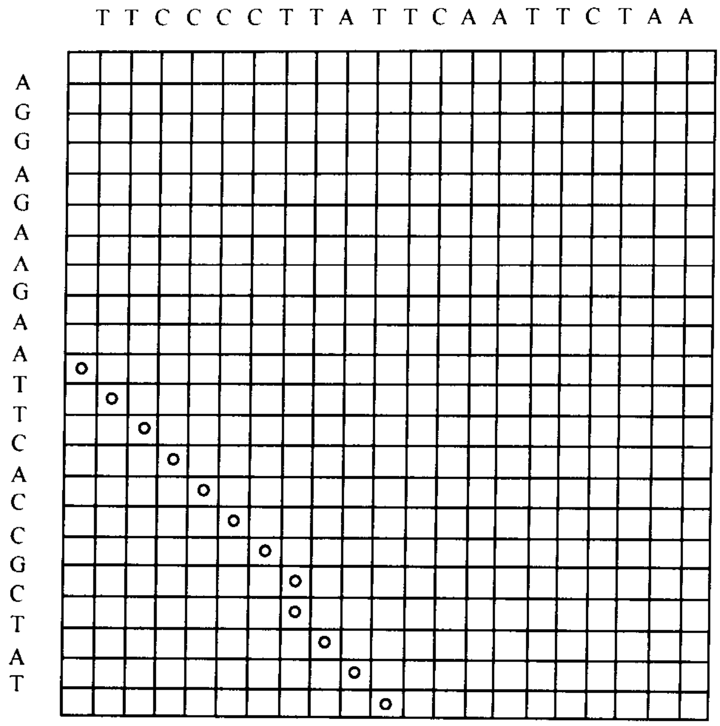


图 4.23　一个优化半全局比对以及它在动态程序设计矩阵中的通路。算法配置为发现前缀－后缀相似性或包含关系，适于片段组装

4.4.2　排序片段

这部分我们简要介绍实际片段组装中排列构造(layout construction)的一个问题，即在一个连叠中发现好的片段排序。给定这样的一个排序，且知道每个片段与下一个片段交叠，则一个排列即可用下一部分提供的方法构造出。但不幸的是，不存在一个既简单又通用的片段排序算法，即使借助于启发式亦如此。因此，与其介绍一个算法，我们不如指出算法中的要点，以便在构造一个好的片段排序时能加以考虑。

在一个实际例子里我们必须处理逆补，因此有必要扩展输入集 $\mathcal{F}$，将片段的逆补加入其中。定义

$$\overline{\mathcal{F}} = \{\overline{f} \mid f \in \mathcal{F}\}$$

和

$$\mathcal{DF} = \overline{\mathcal{F}} \cup \mathcal{F}$$ 136

在一个实际例子里我们还不得不考虑错误，这意味着只能进行大致的匹配。我们将用与先前略为不同的意义使用 $f \to g$，它表示 f 的尾端大致与 g 的头端相同。我们同时假定当 $f \to g$ 时亦有 $\overline{g} \to \overline{f}$，这是合理的，因为无论我们用什么标准处理两个序列，我们应可用同样的标准处理它们的逆补。正如这个例子所示，如果 $f \to g$，且 $f = uv$，$g = wx$，且 v 和 w 是相似的，则这意味着 $\overline{g} = \overline{x}\overline{w}$，$\overline{f} = \overline{v}\overline{u}$，$\overline{w}$ 和 $\overline{v}$ 也是相似的，因此有 $\overline{g} \to \overline{f}$。我们将使用记号

$f \subseteq g$表示 f 是 g 的一个近似的子串。类似于上面的结果，我们同样有 $f \subseteq g \Rightarrow \bar{f} \subseteq \bar{g}$。

我们感兴趣的图是 $\mathcal{OG}(\mathcal{DF})$，其边表示近似的交叠。此外，知道哪些串是其他串的近似子串也很重要。

发现交叠片段的一个好的排序本质上等价于在图 $\mathcal{OG}(\mathcal{DF})$中发现一个有向通路，这些通路产生连叠。注意，对图中的任何通路

$$f_1 \rightarrow f_2 \rightarrow \cdots \rightarrow f_k$$

存在着一个对应的"补"通路

$$\overline{f_k} \rightarrow \overline{f_{k-1}} \rightarrow \cdots \rightarrow \overline{f_1}$$

这个补通路给出了一个连叠，它是原通路的逆补。这意味着目标 DNA 的两条链同时被构造。我们希望得到一对互补的通路，使得每一个非包含片段(non-contained fragment)恰属于其中一个通路。包含片段(contained fragment)可随后定位，因此它们在通路中的出现并非关键性的。实际上，强迫让它们出现反而可能影响通路构造(见习题 14)。只在极少的例子里包含片段可提供连锁，因此在构造通路时我们把它们看做可选部分。

缺乏覆盖和重复片段是两个妨碍构造这对通路的因素，理想情况是这对通路覆盖所有的非包含片段。一个不相连的交叠图通常是缺乏覆盖的表现。如果覆盖良好，重复片段导致交叠图产生环，或在交叠图中产生形如 $f \rightarrow g$ 和 $f \rightarrow h$ 的边，g 和 h 不相关(如果 $g \nrightarrow h$，$h \nrightarrow g$，$g \not\subseteq h$，$h \not\subseteq g$，则 g 和 h 不相关。当片段 f 接近重复段 X 的尾端而片段 g 和 h 是目标序列中使 X 区别于两侧序列的片段时，就会发生这种情形。对于一个确定的片段 f，一个同时包含 f 和 $\bar{f}$ 的环表示一个逆重复段存在于通路中。

重复区域的另一个表现是异常的高覆盖(见图 4.13)。一个重复片段的所有拷贝叠加在一起，导致覆盖增加到平均值与重复片段拷贝的乘积(假定片段在目标序列中大致均衡分布)。

总之，在构造通路时有 4 个问题：①每个通路有一个对应的补通路；②并无必要包括
137 包含片段；③环通常表示出现了重复片段；④不均衡的覆盖也与重复片段有关。

4.4.3 比对与表决

在这一部分大致讨论由交叠图通路构造排列的问题。如果交叠是确切的，则这个任务是直截了当的。如果交叠只是大致的，则需额外注意。我们这里介绍两个与比对构造有关的技术，第一个帮助我们由带错误的通路构造一个好排列，第二个用于对已建立的排列进行局部改进。我们用下面的例子说明第一个技术。

例 4.7 假定我们有一个通路 $f \rightarrow g \rightarrow h$，

f = C A T A G T C
g = T A A C T A T
h = A G A C T A T C C

f 和 g 之间的两个优化半全局比对是

```
CATAGTC---      CATAGTC---
--TAA-CTAT      --TA-ACTAT
```

我们没有任何理由偏好其中的某一个，因为它们的计分是完全一样的。然而，序列 h

加入时，问题就清楚了，即第二个比对更好：

```
CATAGTC-----
--TA-ACTAT--
---AGACTATCC
------------
CATAGACTATCC
```

这个表决 S 满足 $d_s(f,S)=1, d_s(g,S)=1, d_s(h,S)=0$。如果我们将 A 与 G 比对，则第六列的表决将产生歧义，因为 T、A 和 C 各出现一次。在表决序列中，这一列的任何其他选择都产生一个更糟的结果，因为 $d_s(f,S)+d_s(g,S)+d_s(h,S)$的和将更大。

为避免这个问题，我们使用一个结构来表示碱基之间有差异的比对，这些碱基或者实际上匹配于其他的碱基，或者是不受限的碱基，如同上例中序列 g 的第二个碱基 A。我们按下面的方法实现这个想法。每个序列表示为一个碱基的链接表，当形成比对时，匹配的碱基融合，构成同一结点。如果一个碱基的匹配状况不能确定，令其保持未匹配，如图 4.24 中序列 f 和 g。然后，随着新序列加入，这些碱基中的某一些将按合适的比对融合。这个结构可被视做一种无环图，遍历一个拓扑序列产生一个排列。把这个技术应用到例子 4.7，我们得到最好的可能的排列。当 h 加入时，从(f 的)结点 G 到(h 的)结点 A 的一 138
个箭头亦被加入，同时加入了额外的属于 h 的碱基。

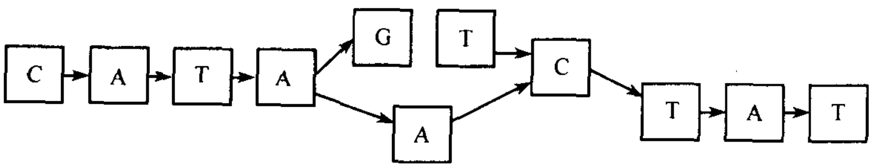


图 4.24 大致交叠的链接结构

```
ACT-GG    ACT-GG
ACTTGG    ACTTGG
AC-TGG    ACT-GG
ACT-GG    ACT-GG
AC-TGG    ACT-GG
------    ------
ACTTGG    ACT-GG
```

图 4.25 同一序列的两个排列。右侧的排列更好，但由于配对比对中的歧义，一个组装程序可能产生左边的排列

我们的第二个技术是改进一个已产生的排列，一个实际中常见的问题见下例。

例 4.8 图 4.25 显示同一序列的两个排列。在左边的排列里，表决序列有两个 T，由多数表决产生，但 5 个片段中的 4 个只包含有一个 T。右边的排列更好，理由是它的表决序列与大多数片段吻合。

解决这类问题的一个办法是执行一个局部性的多序列比对。我们选择一个需要修补的小范围的连续列，从每个片段中抽取出这一部分，然后比对这一部分。对于中等程度的局部覆盖和抽取范围，这个多序列比对可在合理的时间内运行完(尽管是指数复杂性的)。

在这里，使用配对和计分机制就足够了。在例 4.8 中，右边的排列能最大化配对和计分，基于的假定如下：如果 $a=b$ 则 $p(a,b)=1$；如果 $a\neq b$ 则 $p(a,b)=0$；$p(a,-)=-1$；
139 $p(-,-)=0$。

小　　结

本章中我们研究了片段组装问题，这是大规模 DNA 测序中遇到的问题之一。简单地说，片段组装就是按正确的顺序和朝向组装片段集合，这个片段集合由对一段长而未知的 DNA 序列测序获得。

我们首先回顾了该问题的生物学意义以及影响输入的生物学因素，然后我们展示了如何将这一问题转化为一个数学问题，但所提出的模型不能解决所有的相关问题。特别是，如何处理重复片段和利用目标序列的长度信息是很困难的事项。所有的模型均假定片段集合是无嵌合片的，也没有载体污染。基于这些模型的算法也作了介绍。

片段组装需要一类特别的多序列比对，它亦称做排列。在排列中我们必须考虑片段朝向、覆盖以及连锁，对终端空格则不罚分。

由于寻找精确的形式化解法的困难性，在实际中人们常使用启发式方法。问题通常被分解为三个阶段：发现片段间的交叠；构造一个排列；计算表决序列。在没有重复片段的情况下，基于这一策略的程序运行的相当好。我们简要介绍了这种启发式。

新的技术，如杂交测序，可能根本性地改变大规模测序的方式，对此亦需要新的算法。

习　　题

1. 假设我们有如下片段

   f_1：ATCCGTTGAAGCCGCGGGC

   f_2：TTAACTCGAGG

   f_3：TTAAGTACTGCCCG

   f_4：ATCTGTGTCGGG

   f_5：CGACTCCCGACACA

   f_6：CACAGATCCGTTGAAGCCGCGGG

   f_7：CTCGAGTTAAGTA

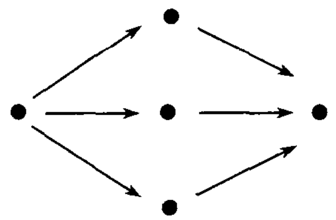
   f_8：CGCGGGCAGTACTT

   而且已知目标序列的总长度大约是 55 碱基对，组装这些序列并产生表决序列。注意错误处理，或许应用到某些序列的逆补。
2. 对 4.1.1 的排列，什么是最小、最大、平均覆盖？
3. 在重构模型里，什么是使图 4.2 中答案有效的最小的 ε 值？
4. 问题同习题 3，材料是图 4.3 和图 4.4 中的排列。
5. 在图 4.8 中，假定存在一个由 B 的端点、X 的一个拷贝以及 D 的起点构成的一个片

段，这个片段如何影响组装？

6. 对 $\mathcal{F}$ = {AAA, TTA, ATA} 构造交叠图，并找出一个最短公共超串。

7. 找出产生下列交叠图的序列，图中只画出了正权值边。具体的权值并不重要，只要它们是正的。



8. 设计一个算法找出两个长为 m 和 n 的串的最大确切交叠，算法的运行时间需为 $O(m+n)$

* 9. 设计一个方法找出 DNA 序列中的大致重复片段。

10. 给出一个 DNA 序列例子，它无法由 8-mers 无歧义地重构。

11. 证实 SCS 算法不能解决 4.1 节中的例子，在每一种情况下最短公共超串是什么？

12. $\mathcal{F}$ = {ATC, TCG, AACG}，根据重构模型（ε 分别等于 0.1 和 0.25），找出这个集合的最佳排列，注意考虑逆补。

13. $\mathcal{F}$ = {TCCCTACTT, AATCCGGTT, GACATCGGT}，根据多连叠模型（$\varepsilon = 0.3$，$t = 5$）找出这个片段集的最佳连叠集合。

14. 给出一个集合 $\mathcal{F}$，使得 $\mathcal{F}$ 的最短超串不对应于 $\mathcal{OG}(\mathcal{F})$ 中的通路。

15. 对每一个 $m > 0$，找出一个无子串集合 $\mathcal{F}_m$，使得贪婪算法返回一个至少比 $\mathcal{F}_m$ 中最短者长 m 个字符的超串。

16. 令 A、B、C、D、E、X、Y 和 Z 是目标 DNA 序列中的块，假定这些块大于目标分子中任何采样的片段。用 X、Y、Z 填下面的空，使得这个序列变成固有的歧义序列。

$$\text{target} = A\ X\ B \ldots C\ Y\ D \ldots E\ Z$$

17. 你如何用 BLAST 或其他数据库搜索工具筛选有载体污染的片段？

18. 找出一个多项式时间的把 SCS 简约为重构模型的算法。

* 19. 假定一个序列有一个交叠的重复片段 S，证明 S 或者形如 XXX，或者形如 $X(YX)^k$（对某些非空块 X 和 Y，以及 $k \geqslant 2$）。

◇ 20. 设计一个算法接收一个 DNA 序列集合，并构造一个长度至多为最短串 2 倍的公共超串。 141

文 献 提 要

最短公共超串问题已被详细研究。Turner[187] 描述了几个算法，包括贪婪算法，并说明了性能。Li 给出了第一个实际长度超串的性能保证[126]，它的上界是 $O(n\log n)$，n 是优化长度。接着，Blum、Jiang、Li、Tromp、Yannakakis 把结论改进到 $3n$[24]，并展示贪婪算法至多在 $4n$ 长时仍能产生超串。这引出了一系列文章使用越来越优化的技术达到越来越好的比率。在这方面，Teng 和 Yao 证明了一个比率 $2+8/9$[185]；Czumaj、Gasienec、Piotrow、Rytter 证明了一个比率 $2+5/6$[41]；Kosaraju、Park、Stein 证明了一个比率 $2+$

50/63[118];Armen 和 Stein 证明了比率 2 + 3/4[15] 和 2 + 2/3[16]。大多数这些算法运行时间为 $O(|S|+n^3)$,这里 $|S|$ 是超串大小,n 是片段数量,见 Gusfield 的工作[84]。用 Gusfield、Landau、Schieber 的方法[86]以及后缀树,图 $\mathcal{OG}(\mathcal{F})$ 可在时间 $\mathcal{O}(\|\mathcal{F}\|+n^2)$ 内被构造。

重构模型出现于 Peltola、Söderlund、Tarhio、Ukkonen 的论文[157]。Kececioglu 的学位论文[114]以及 Kececioglu 与 Myers 的论文[115]是基于重构模型的,多连叠模型亦由他们提出。根据在排列中片段端点的分布,Myers 提出了一个片段组装形式化的有力方法[144],特别适用于包含重复片段的例子。基于一类特殊的不同于本书提到的图,他还提出了一个算法,图 4.12 即基于论文中出现的一个例子。

几个课题组实现了 DNA 片段组装的程序,早期的系统可见于一些文献[71,138]。一个试图集成合理的数学形式化与实际的应用的努力来自 Peltola、Söderlund、Ukkonen[158]。Huang[98]给出了一个成功的片段组装程序,改进后可处理重复片段、检测嵌合片段、自动编辑排列[99]。Kececioglu 和 Myers 把整个过程分隔为三个阶段[115],提供了详细的形式模型以及用于每一阶段的确切和近似的算法。Staden 自 70 年代末以来维护了一个具有很强片段组装能力的序列处理系统[178,46]。非常流行的 GCG 系统也提供片段组装支持[47]。片段组装系统的设计目标由 Myers 作了总结[143]。一个好的关于自动化 DNA 处理的一般文献索引是 Adams、Fields、Venter 的书[1],本书中 4.4.3 的技术细节的讨论亦可见于 Meidanis 和 Setubal 的论文[137]。

Engle 和 Burks 描述了一组用于产生片段组装的人工实例的工具[53,54]。有一个实际测序项目的数据集,这些数据可作为 Seto、Koop、Hood 开发的计算机化的组装工具的测试例[171]。方程(4.1)和(4.2)描述了一个霰弹枪法测序项目的进展,这个项目由 Waterman 报道[199]。这一论题的早期工作由 Lander 和 Waterman 做出[120]。

142 Waterman[199]描述了一个 SBH 算法,并介绍了一篇更具体的论文。

片段组装是 NSF 离散数学与理论计算机科学中心在 1995 年座谈会的主题之一,有关那次座谈会的一本论文集即将面世。关于座谈会的更多细节见

http://dimacs.rutgers.edu/SpecialYears/1994_1995/challenge.html

第 5 章　DNA 物理作图

正如我们在第一章中所见，一条人类染色体是一个长约 10^8bp 的 DNA 分子，而迄今为止用于 DNA 序列测定的技术仅限于长度为几万个碱基对的 DNA 片段，这意味着测序的 DNA 片段总是染色体的极小的一部分。这就像我们用一个精细望远镜从月球上观察地球，而这个望远镜仅适于在不大于 1 英里的视野内(相当于15 000bp 的 DNA 片段的长度)分辨相隔几英寸的物体(相当于一个碱基，图 1.10)，因此这种望远镜当然不是研究山脉、大洲和海岛等大尺度结构的合适工具。实际上，要观察这些结构需要其他工具。与之类似，分子生物学家需要使用特殊的技术来处理规模上与染色体具可比性的 DNA 分子，这些技术使他们能够制成整条染色体或其主要部分的图谱。本章我们要讨论的是在制备物理图谱过程中对生物学家有帮助的计算技术。

5.1　生物学背景

一个 DNA 片段的物理图谱告诉我们 DNA 分子上特定的标记(marker)所在的位置。 143
这些标记通常很小，它们是精确定义的序列。这样的图谱可帮助分子生物学家进一步研究基因组。例如，假定一段 DNA 序列已被完全测序，称为序列 S。若已知序列 S 来自哪条染色体，且我们有该染色体的物理图谱，我们就能试着找出图谱的位于 S 区域的某个标志。如果找到，则表明我们成功得到了序列 S 在该染色体中的定位(见图 5.1 说明)。 144

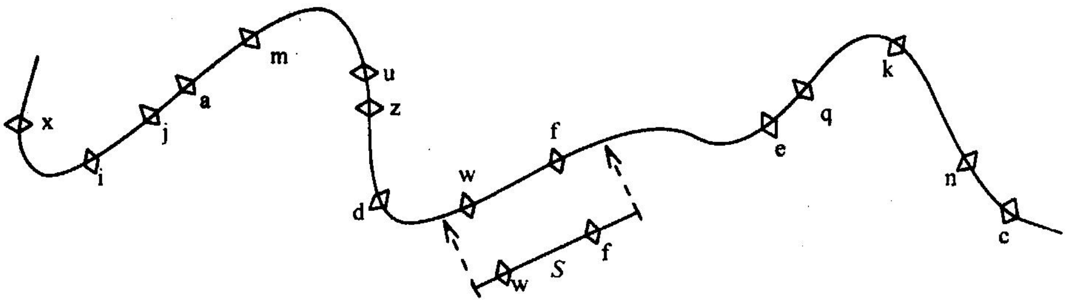


图 5.1　物理图谱的略图。该图中非常长的 DNA 分子(例如染色体)以实曲线表示，沿着分子的标记以菱形方块表示，每一个标记相互区别，由与之对应的字母来表示。片段 S 表示已测定序列的 DNA 片段，例如，它可包含一个基因。根据物理图谱我们可确定 S 在该染色体上的位置

那么接下来的问题是，如何绘制物理图谱？首要的任务是要得到许多拟作图 DNA 分子(目标 DNA)的拷贝，用限制酶将每一拷贝切成几个片段(有关 DNA 片段限制酶裂解的描述见 1.5.2)。作图的依据来自对这些片段的仔细比较，尤其是观察交叠。需要强调的是，通常这些片段仍然太长而不能直接进行测序，因此我们不能像第 4 章谈到的片段组装那样通过测序和片段比较来确定交叠，而是要通过产生这些片段的 DNA 指纹(fingerprint)来获得交叠信息。正如我们的指纹能惟一地标识身体的某个部分，DNA 指纹能以独特的方法描述一段 DNA 中所包含的特定信息。两种常见的制备 DNA 指纹的方法分别是限制酶酶切位点分析(restriction site analysis)和分子杂交(hybridization)。

限制位点作图的目的是对目标 DNA 分子内某种给定的限制酶的作用位点进行定位，这些位点由此成为前面提及的标记。定位技术基于对酶切片段长度的测量，因此一个片段的长度就是它的指纹。在杂交标图中，我们检测某些小序列能否与酶切片段结合，那些能与酶切片段结合的小序列子集就是指纹。DNA 指纹可通过杂交信息加上片段长度而获得，但为了阐述清楚，在本章中我们分开讨论这两种技术。在任何情况下，通过比较 DNA 指纹，我们可试图确定片段间的交叠关系，进而确定它们之间的相对次序(这种次序信息在 DNA 打断的过程中丢失了)。

如同分子生物学中遇到的其他问题一样，信息的缺乏和大量实验错误的存在使得物理标图问题尤其艰难。有时，根据给定的片段集获得一个连续的物理图谱甚至是不可能的。仅仅由于酶切过程未能产生覆盖目标 DNA 某些特定区域的片段，这种情况就可以发生。此时，我们称这些不连续的物理图谱片为**连叠群**(contigs，contig 的复数，我们在片
145 段组装中采用了同样的名字)，这是所有物理标图过程共有的特征。下面分别详述以上所提及的技术。

5.1.1 限制位点作图

本节我们简要介绍两种作图技术，它们基于对限制位点之间的片段长度的测量。

获得大量大小不同的 DNA 片段的方法之一是使用两种而不是一种限制酶对目标 DNA 进行消化。我们将举例解释这种技术。假定有限制酶 A 和 B，每种酶识别不同的序列(限制位点)，先用酶 A 消化目标 DNA 分子，获得一份片段集，其长度分别为 3、6、8 和 10 个长度单位(长度单位可以是几千个碱基对)；再用酶 B 消化目标 DNA 分子，获得另一份片段集，其长度分别为 4、5、7 和 11 个长度单位；最后再用酶 A 和 B 联合消化目标 DNA 分子，得到第三份片段集，长度分别为 1、2、3、3、5、6 和 7 个长度单位。现在的问题是如何对这些片段进行排序使得它们的顺序与实验结果一致？我们称这类问题为*双重消化问题*(double digest problem)，因为酶对分子链的裂解过程称为消化。对这一具体例子，一个解决办法如图 5.2 所示。

双重消化的一种变体是*部分消化*(partial digest)，这时只采用一种限制酶，但进行多次实验来消化目标分子的许多拷贝，得到作用于不同拷贝的不同的反应时间。反应时间的长或短决定了被识别的限制位点的多或少，因而产生不同长度的片段。理想情况下，这种实验对每一对限制位点至少可提供一个片段。接着就可以通过分析片段的长度来定位限制位点，如图 5.2 中酶 A 可识别 3 个限制位点，该酶的部分消化实验可产生如下长度

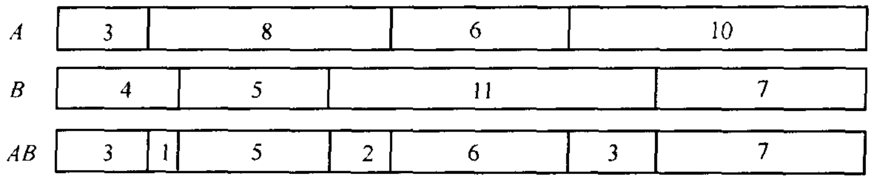


图 5.2　双重消化问题的解决办法

的片段集合:3、11、17 和 27(左端点与所有其他识别位点之间的片段);8、14 和 24(第一个限制位点与其右侧所有位点间的片段);6 和 16(第二个限制位点与其右侧所有位点之间的片段);10(最后一个片段)。

消化数据可能会有以下几种实验错误。首先,是片段长度测定的不确定性,片段的长度通常通过凝胶电泳来检测,因而其误差率可达 5%;其次,如果片段太小的话,我们根本
不可能检测它的长度;第三,有些片段还可能在消化过程中丢失了,这就导致了 DNA 分 146
子覆盖之间存在空隙。所有这些错误都使标图过程变得更加复杂。即使这些问题并不存在,我们随后将看到有关的计算问题也相当困难。

5.1.2　杂交作图

回顾先前对杂交作图的讨论,片段间交叠的信息来自于每个片段所含内容的部分信息。在解释什么是部分信息之前,必须强调在本部分里我们称片段为**克隆**(clone),因为借用了一种叫做克隆(cloning)的技术(见 1.5.2)。在用不同的方式将每份目标 DNA 分子打碎并将片段克隆之后,我们就得到了几千个克隆构成的集合,称做*克隆库*(clone library),每个克隆的长度通常为几千个碱基对。

每个克隆的部分信息是通过*杂交实验*(hybridization experiment)来获得的。在这些实验里,我们检测被称为**探针**(probe)的小序列可否与克隆结合(或杂交,hybridize)。如果发生了结合,就说明该克隆包含与该探针序列互补的序列。对于同一个克隆,我们可采用不同的探针重复杂交实验,与该克隆杂交的所有探针的集合称为该克隆的指纹。指纹重叠的两个克隆可能来自目标 DNA 分子的交叠区域。例如,如果我们知道了探针 x、y 和 z 能与克隆 A 结合,而探针 x、w 和 z 能与克隆 B 结合,我们就很有理由相信克隆 A 和 B 互相交叠,如图 5.3 所示(除非存在重复片段,即 repeats,我们将随后讨论)。注意,这类信

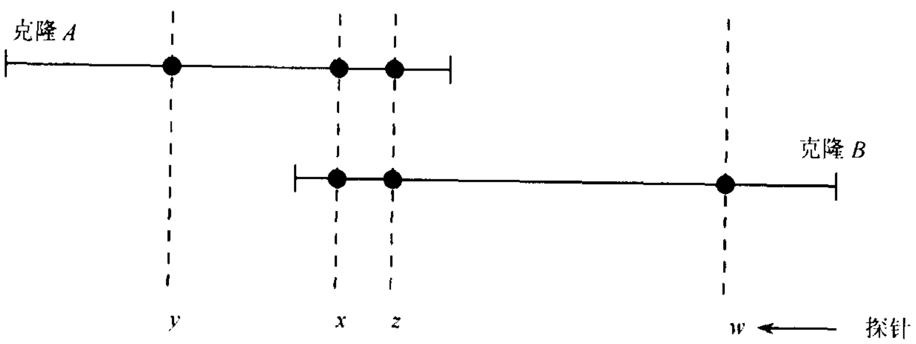


图 5.3　4 个探针如何与 2 个克隆杂交的例子

息一般并不能足以使我们确定这些探针在目标 DNA 中的位置，而仅能确定它们的相对顺序。我们后面将进一步讨论这点。

在杂交实验中各种错误都可能发生。首先，探针可能结合失败，这就产生了**假阴性**(false negative)的结果；其次，探针也可能结合到了不该结合的位置上，产生**假阳性**(false
147 positive)结果；有时还会出现人为对实验结果的判读错误，这也可能产生假阳性或假阴性结果。有时，甚至在杂交尚未发生前错误就已出现。在克隆过程中，DNA 分子的两个片段可能会发生融合并像单一克隆那样进行复制，这种克隆称之为**嵌合克隆**(chimeric clone)，它可使人们产生关于探针相对顺序的错误推断。嵌合现象常有发生，故其已成为杂交标图中的一个严重问题。有估计认为在许多克隆库中有高达 40%～60%的克隆实际上是嵌合体。克隆过程中的另一类错误是**缺失**(deletion)，即克隆丢失了一个内部片段，因而使目标 DNA 分子中两个本不相邻的片段连接在一起。

另外还有两种情况也可使杂交标图过程出现问题。第一是探针不惟一，这意味着探针可与目标 DNA 分子一个以上的位点结合，这些位点序列称为*重复序列*(repeat)。一种称做*序列标记位点*(sequence tagged site, STS)的技术可避免这个问题，它产生的探针可与目标 DNA 分子上专一的位点进行杂交。第二个问题是数据缺乏，因为在杂交实验中完成所有的杂交不太可行。*分享技术*(pooling technique)是解决此类问题的措施。

5.2 模型

介绍了各种各样的标图策略后，我们在本节综述与之对应的各种数学模型，以探索这些模型所衍生出来的标图问题的计算复杂性。下面将会看到，标图问题在算法上是困难的，但它亦引出了问题求解方面的几个有趣问题，在本节末我们将进行讨论。

5.2.1 限制位点模型

我们先看看简称为 DDP(double digest problem)的双重消化问题的模型。在该模型中，每个片段以其长度来表征，并且假定消化过程完美无缺，即对长度的测量不存在误差，而且所有的片段都存在。这显然是不现实的，但即便过分简化至此，计算问题仍然非常困难。

将实验结果置入模型的框架，每一个实验都可得到一个长度的集合。如第一种酶消化目标 DNA 分子后得到集合 $A=\{a_1,a_2,\dots,a_n\}$，集合中的某些片段长度可能相等，这意味着该集合实际上是一个*多重集*(multiset)。与此相似，第二个酶可得到集合 $B=\{b_1,b_2,\dots,b_m\}$。最后用酶 A 和 B 双重消化后得到多重集 $O=\{o_1,o_2,\dots,o_k\}$。

我们要从这些给定的多重集里发现什么呢？我们需要 A 中元素的一个置换 π_A 与 B 中元素的一个置换 π_B，以便获得下面的结果：根据 π_A 给出的次序，在一条线上画出来自
148 A 中元素的长度；类似地，根据 π_B 给出的次序在线上画出来自 B 中元素的长度。由于交叠，我们现在有几个新的子区间。现在有可能建立所产生子区间和 O 中的每个元素之间的一一对应关系。如果能做到这一点，则我们说 π_A 和 π_B 是 DPP 的解。

找出这些置换是一个 NP-完全问题，这里给出其证明。首先容易看出，对一个给定的

DDP 解，很容易检验它是否为真实解，所需做的是排序 O 集合并画出子区间，以确定是否可获得同样的结果。为完成这个证明，我们注意到 DDP 仅是集合分划问题的一个推广，而集合分划问题是一个众所周知的 NP-完全问题。

在集合分划问题中，对给定整数集 $X=\{x_1,x_2,\ldots,x_l\}$，我们想知道是否可把 X 分成 X_1 和 X_2，使得 X_1 中的元素和与 X_2 中元素的和相等。这对应于一个双重消化问题，其中 $A=X, B=\{K/2,K/2\}, O=A$，这里 $K=\sum_{x\in X}x_i$。换句话说，集合分划问题是一个 DDP 问题，其中一个酶仅产生两个片段，二者具有相同的长度。

由于我们总是寻找 DDP 的真实解，这个问题便有了另一面，它大大增加了搜索的难度：解的数量是呈指数增长的。为进一步理清这个问题，考虑这种情况，即被酶 A 所切的两个位点之间存在被酶 B 切割的三个位点 b_1、b_2、b_3，在这种情况下我们将无法知道片段 $[b_1,b_2]$ 和 $[b_2,b_3]$ 的相对次序。通常，对于一个限制酶的 k 个位点，我们可有 $(k-1)!$ 个解。实际上，情况因重合的出现会变的更糟。当酶 A 的某个位点和酶 B 的某个位点非常接近时，便产生了重合(它们不会完全重合，为什么?)。当出现连续的重合时，A 酶与 B 酶的对应片段将看起来在长度上相同(由于测量的不准确性)。所以，n 个连续的重合将导致 $n!$ 种可能的解。进一步讨论可知，解的数量有可能随目标 DNA 的长度呈指数增加(见文献提要)。

所以我们可以看到，即便一个小小的双酶切问题也可造成严重的计算问题。部分酶切的 NP-完全性尚未被证明。因下列原因，它看上去更容易处理。在部分酶切法标图中，有 n 个酶切位点(包括 2 个末端)，在实验中，对每一对酶切位点我们至少可获得一个片段，或者说总共有 $\binom{n}{2}$ 个片段。测定这些片段的长度后产生一个距离的多重集。问题是确定这些酶切位点的位置，使得位置间距离的集合与从酶切消化所获得的片段长度一致。这个问题之所以略为简单，是因为研究者已发现解的最大数量并不大，因此从组合学的观点来看部分消化是双重消化的一种好的替代。 149

5.2.2 区间图模型

使用简单的基于区间图(见 2.2 节的定义)的模型，我们可初步考察杂交标图(或更一般地，指纹标图(fingerprinting mapping)的计算复杂性。这些模型抽象了标图的许多细节。然而，尽管简单，如同先前所见的双重消化一样，它们也导致 NP-难题。

在这些模型中，我们创建图，图的顶点代表克隆，图的边代表克隆之间的交叠信息。如果我们完成图并确保有关克隆交叠信息的正确，所得到的图就是区间图，这是这些模型的一个重要特征。

在第一个模型中，我们创建两个图。第一个图是 $G_r=(V,E_r)$。如果 $(i,j)\in E_r$，就意味着我们确信克隆 i 和 j 交叠。第二个图是 $G_t=(V,E_t)$，其中 E_t 代表已知和未知的交叠信息(即 $E_r\subseteq E_t$)。G_t 不必是完全的，因为我们可以确知某些克隆对并不交叠，因此对应的边在 E_t 中被省略。我们要解决的问题是：是否存在图 $G_s=(V,E_s)$，使得 $E_r\subseteq E_s\subseteq E_t$，且 G_s 是区间图？注意，G_r 和 G_t 并不要求是区间图(在这种情况下解是平凡的)。见图 5.4、图 5.5 和图 5.6 中的例子。

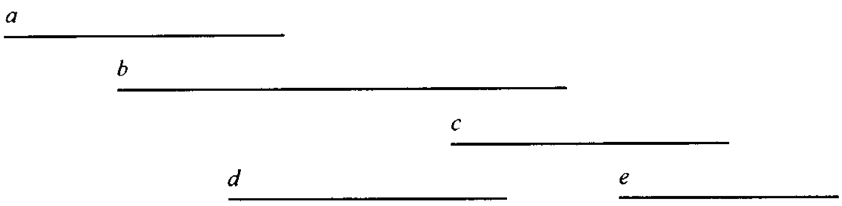


图 5.4　克隆交叠模式的例子

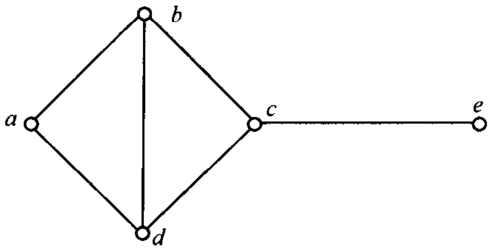
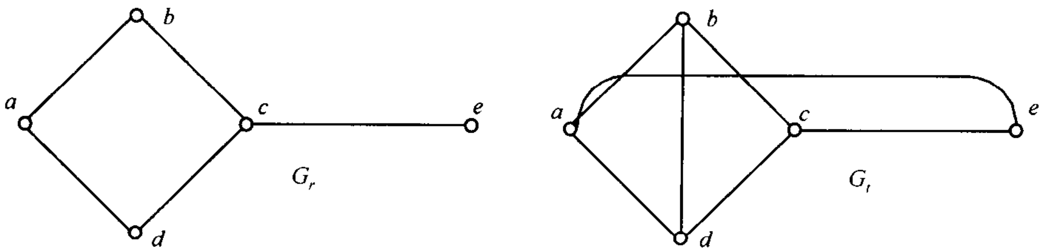


图 5.5　对应于图 5.4 的区间图



150 图 5.6　对图 5.4 的例子，可能的图 G_r 和 G_t，二者均不是区间图

在第二个模型中，我们假定已知的重叠信息不是可靠的，但仍应用有关的信息构建图 $G=(V,E)$，需要解决的问题是：是否存在图 $G'=(V,E')$，使得 $E'\subseteq E$，G'是区间图，且 $|E'|$的值最大？对 $E'\subseteq E$ 的要求意味着，若想获得这个问题的一个解，我们也许不得不丢弃 G 的一些被解释为假阳性的边。问题的解是假阳性数量最少的解释。

在第三个模型中，我们同时利用交叠的信息及每一个克隆来源的信息。由于克隆来自同一分子的不同拷贝，一个合理的想法是用克隆所属分子的分子标志来标记每一个克隆。假定我们有目标分子的 k 个拷贝，用不同的限制酶来打断不同的拷贝。现以克隆间的交叠信息构建图 $G=(V,E)$。因为克隆已标记过，我们可将对应的顶点以 k 种颜色着色（一种颜色对应一个拷贝）。在所构建的图中，同种颜色的顶点间不会有边连接，因为它们对应于来自同一分子拷贝的克隆，因此不会产生交叠，我们这时说该图有一个有效着色。需要解决的问题是：是否存在图 $G'=(V,E')$，使得 $E\subseteq E'$，G'是区间图，且 G 的着色对 G'亦有效？换言之，能否给 G 增加边使之在不损坏着色的条件下转化为区间图？

刚才我们描述了杂交标图的三种不同模型，都比较简单，可是所描述的三个问题都是 NP-难题，此处不给出证明，读者可在本章后的文献提要中找到。以下我们继续描述更特定的模型。

5.2.3　连续 1 特性

前面描述的模型适用于杂交作图，但在原理上它们可应用于任何情况，只要我们能获得每一片段的某种指纹。现在介绍一个模型，使用与克隆结合的探针集合作为克隆指纹。在这个模型中我们做下列假设： 151

- 每一个探针的互补序列在目标 DNA 上仅出现一次(探针是惟一的)；
- 不存在错误；
- 所有“克隆 × 探针”杂交实验已完成。

读者可能会认为这会是另一个 NP-难题，仅此一次情况并非如此。我们将会看到所引发的问题是多项式时间可解的，虽然其某些一般化的形式确实是 NP-难题。

假定有 n 个克隆和 m 个探针，实验数据能使我们构建 $n \times m$ 二元矩阵 M，其中入口项 M_{ij} 告诉我们探针 j 是与克隆 i 杂交($M_{ij}=1$)或不杂交($M_{ij}=0$)。从这个矩阵中获得一个物理图谱便是一个发现置换问题，这个置换包含所有的列(探针)，且每一行中所有的 1 都是连续的。能够发现一个置换的这样一个二进制矩阵被称为具有行的连续 1 特性(consecutive 1s property)，或简称为 C1P。以下我们直接使用术语 C1P 置换和 C1P 问题。

检验一个矩阵是否具有这个特性并发现一个有效的置换是一个众所周知的存在多项式算法的问题，我们将在 5.3 节给出一个算法。但是，这个模型明显过于简单。如果存在错误，即便在真正的置换中一些行中的 1 也可能不是连续的。怎么办？回答是：审视无错的例子可帮助我们获得直觉以设计对含错数据亦有效的算法。特别是我们可使用下面的重要假设：实验室研究人员尽力减少错误的发生。如果实验是完美的，所获得的杂交矩阵将存在 C1P，这意味着任何旨在试图找到或逼近真实列置换的算法都可产生解，这个解可将可能由实验错误所引发的问题减小到最少。因此，这样的算法能够找出 C1P 置换，如果 C1P 置换存在的话。

值得注意的是，即便存在一个 C1P 置换，我们也不能说它就是真正的置换。首先，对一个给定的矩阵可以存在几个这样的 C1P 置换，如何知道哪一个是真的呢？一般性的结论是：我们无从知道，我们仅能说所有的 C1P 置换都是真正置换的候选。基于这个原因，针对 C1P 问题的算法应能给出不止一个解，而是全部解。另外，若错误存在，甚至可出现这种情况，如前面所提到的，即便存在一个 C1P 置换，真正的置换也可以不是 C1P 置换。

另一方面，如何改进模型以容忍错误？有多种可能。与其要求每行恰有一个连续 1 的块，不如试图寻找一个列置换，使得每一行至多有 k 个连续 1 的块，k 的值可以是 2 或 3。我们亦可试图将矩阵中连续 1 的总块数最少化，这样，我们可指望大多数行将恰有 1 个连续 1 的块，偶尔一行可有 4 到 5 个。不幸的是，正如前面所暗示的，这种一般化会产生 NP-难题(见习题 16)。如果放宽探针必须是惟一的假设，即便不存在错误，NP-难题也会产生(参见文献提要)。 152

5.2.4　算法启示

在描述用于获得物理图谱的算法技术之前，我们初步探讨问题的困难性给我们带来的限制，这些讨论揭示了怎样算是用算法“解决”了问题，因此其意义不局限于物理作图

问题。

第一个考虑是,我们所要解决的是现实生活中的实际问题而不是抽象的数学问题。实际问题是这样的:我们要知道克隆在目标 DNA 中的真实次序。这是一个组合数学问题,存在着许多可能的次序,而仅有一个是我们需要的。

我们试图通过使用处理优化问题的抽象模型来发现真实次序。这些问题本身就可能非常困难,更不用说它们是更困难的实际问题的抽象。例如,我们在 5.2.2 所看到的所有问题的优化版本均是 NP-难题。通常我们无法快速地找到这些问题的优化解。即便我们在合理的时间内最优地解决了其中某个问题,我们也不能保证所得到的解对应于真实的解,因此我们似乎处于可怕的困境中。

尽管如此,情况并不像看上去那么暗淡。首先,既然我们不能保证任何优化解是真实的解,好的近似解应同样能满足我们的目标;第二,我们须明了分子生物学的算法是反复使用的,科学家获得实验数据并输入到算法中,基于算法给出的解,科学家可以进行更多的实验以获得更多的数据,并重新使用算法。每一步我们都可期望发现更好的(或更接近真实的)解。

这些事实提示了标图算法的必要特征,我们列出如下:

- 假定错误率保持不变,应用更多的数据时,算法能执行更好。
- 它应能呈示蕴含解的丰富的细节框架,特别地,呈示解是如何获得的,甄别解中"好"的(存在强烈的次序征兆的克隆群)与"坏"的部分,这将大大简化进一步的实验。
- 如果几个候选解均符合优化条件,算法应能报告出全部候选解。若报出了过多的解,则说明优化条件太弱(或者输入的数据包含太多的错误);反之,如果没有给出解,则表明优化条件太强。

我们希望存在这样的保证条件,即真实的解存在于由算法给出的近似或优化解中。一般地,这是不可能的。取而代之的是,我们试图获得几个不同的优化函数以及相应的算法,它们的最优值享有真实解的某些特性,目的是希望真实解存在于由算法所得到的解的交集中。在 5.4.2 我们给出这种保证的一个例子。

注意,不同于使用上面介绍的几个算法,我们也可尝试设计能够优化多个目标函数的
153 算法,这是一个非常重要的研究领域。

5.3　一个 C1P 问题的算法

在本节中,我们介绍一个简单的算法,它能确定一个 $n \times m$ 二进制矩阵的行中是否有 C1P。算法的目标是找到一个列的置换,使得每行的所有 1 连续。为简化问题,假定全部的行都不同(也就是说,没有两个克隆具有同样的指纹),同时不存在全是 0 的行(即每一个克隆至少与一个探针杂交)。

要理解这个算法,首先需观察矩阵行之间相互关联的方法,为此需要下面的定义。

定义 5.1　对 M 的每一行 i,令 S_i 为 k 列的集合,这里 $M_{i,k}=1$。

根据这个定义,我们可把一行与另一行关联。给定两行 i 和 j,可出现三种情况:

1. $S_i \cap S_j = \phi$;

2. $S_i \subseteq S_j$ 或 $S_j \subseteq S_i$;

3. $S_i \cap S_j \neq \phi$,二者互不为子集。

我们可清楚地分别处理符合第一种情况的各行,因为对应于 S_i 中的元素的列的置换不会干扰 S_j 中的元素。在这种情况下,我们可把 i 和 j 放入不同的组分且分别处理这些组分。那么,行之间有非空的交集情形如何(此处我们用词不严谨,以行来代替对应的集合 S)? 它们是否应在同一组分中? 我们仔细考察这种情况。

让我们先把所有有非空交集的行归于一个组分中。假定组分中存在一个行 k,使得 S_k 或是 S_i 的子集或与 S_i 不相交,所有的 $i \neq k$。显然,在处理这个组分时可以不管行 k(以及其他具有同样特性的行),行 k 将在自己的组分中,或与其他行共享组分,至于这些其他行是什么将随后见分晓。

基于给出的直观分析,我们给出解决两行是否属于同一组分的形式解法。用矩阵 M 构建图 G_C,G_C 中的每一个顶点将是 M 中的一行。如果 $S_i \cap S_j \neq \phi$ 且二者互不为子集,则从顶点 i 到 j 有一无向边,我们想要的组分将是 G_C 的连通成分。作为一个例子,图 5.7 显示了对应表 5.1 矩阵的图 G_C。

表 5.1 一个二进制矩阵

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
l_1	1	1	0	1	1	0	1	0	1
l_2	0	1	1	1	1	1	1	1	1
l_3	0	1	0	1	1	0	1	0	1
l_4	0	0	1	0	0	0	0	1	0
l_5	0	0	1	0	0	1	0	0	0
l_6	0	0	0	1	0	0	1	0	0
l_7	0	1	0	0	0	0	1	0	0
l_8	0	0	0	1	1	0	0	0	1

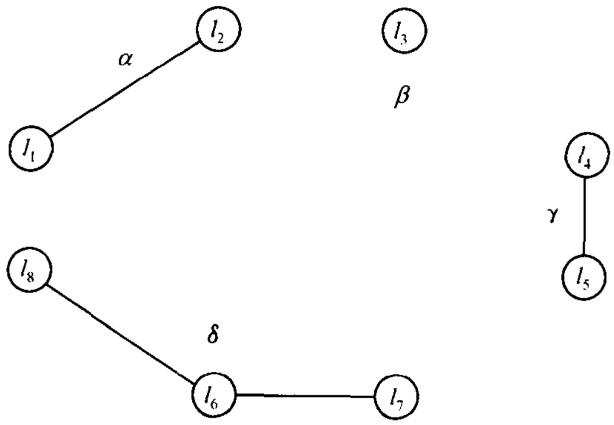


图 5.7 图 G_C 对应表 5.1 的矩阵,以希腊字母表示连通的组分

现在我们可以勾勒出该算法。按照上面的规则将行分成不同组分,置换每一组分的列,然后以某种方式将组分汇合起来。这留给我们两个子问题:如何发现一个组分的正确置换,以及如何将组分汇合起来。让我们首先设计组分内的列置换。 154

=========== 组 分 处 理 ===========

为解释算法的这一部分,我们将利用表 5.2 所示的矩阵例子(假定其仅仅是某个的矩阵的一部分)。

让我们首先看一看行 l_1。如果它是这个组分的惟一行,我们所需做的就是置换列使得全部的 1 变为连续的。值得注意的是,若该行中有 k 个 1,则将有 $k!/2$ 个置换。我们不把置换的反转算作不同的解,因为其对 1 的连续性并无影响。本例中行 l_1 的 1 在列 2、

7、8,那么可能置换的集合是 2、7、8,或者 7、2、8,或者 2、8、7。我们可以编码所有这些解,在一个给定行中把列的集合编号为 1(我们称这些集合为列集合)。对于一行,这个集合
155 就是上面定义的集合 S,其结果如下:

表 5.2　二进制矩阵的一部分

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
l_1	0	1	0	0	0	0	1	1
l_2	0	1	0	0	1	0	1	0
l_3	1	0	0	1	0	0	1	1

		{2,7,8}	{2,7,8}	{2,7,8}	
$l_1 \rightarrow \cdots$	0	1	1	1	0 ⋯

现在,让我们观察行 l_2。因为该行与 l_1 在同一组分中,必然有这样的情况:对某些列 l_2 有 1 而 l_1 亦有 1,而对另一些列 l_2 有 1 但 l_1 则没有。现在必须用与 l_1 一致的方式放置 l_2,我们有两个选择:把 l_2 的 1 置于 l_1 的 1 的左侧或右侧,这意味着列 5 在列 2 和 7 的左侧或右侧。假定放置列 5 于左侧,用上述列集合编码的可能的解便是:

		{5}	{2,7}	{2,7}	{8}	
$l_1 \rightarrow \cdots$	0	0	1	1	1	0 ⋯
$l_2 \rightarrow \cdots$	0	1	1	1	0	0 ⋯

如果在这个组分中仅有两行,我们将得到其列的两种可能置换:5、2、7、8 或 5、7、2、8。如果置列 5 于右侧,则得到这些置换的反转,这意味着第二行的放置方向对求解没有影响,这也是一个关键性的观察结果。

现在我们考虑行 l_3。因为它与其他两列在同一组分中,在图 G_C 中应能找到边(l_3, l_1),或边(l_3, l_2),或二者均能找到。现在根据 l_2 放置 l_3,看起来好像要么把 l_3 放置在 l_2 的右边,要么放置在左边,但必须同时考虑 l_3 和 l_1 之间的关系。可通过考虑 S_1、S_2、S_3 交集的元素个数来决定。对任意两个集合 x 和 y,令 $x \cdot y = |S_x \cap S_y|$(行 x 和 y 的内积)。如果 $l_1 \cdot l_3 < \min(l_1 \cdot l_2, l_2 \cdot l_3)$,则行 l_3 与 l_1 的方向须与 l_2 与 l_1 的方向相同;而若 $l_1 \cdot l_3 > \min(l_1 \cdot l_2, l_2 \cdot l_3)$,则行 l_3 与 l_1 的方向须与 l_2 与 l_1 的方向相反(不能有相等,为什么?)。在这两种情况下,若组分是 C1P,则 l_1 和 l_3 必然放置得合适;反之,若 l_1 和 l_3 放置不合适,则组分不是 C1P。原因是基于这样的事实:我们迄今作出的惟一选择是 l_2 相对于 l_1 的放置,且两种可能性都导致同样的解。

在上述的例子中,$S_3 = \{1,4,7,8\}$,那么 $1 \cdot 3 = 2, 1 \cdot 2 = 2, 3 \cdot 2 = 1$,这意味着我们必须
156 把 l_3 置于 l_2 的右侧,如下图所示:

		{5}	{2}	{7}	{8}	{1,4}	{1,4}	
$l_1 \rightarrow \cdots$	0	0	1	1	1	0	0	0 ⋯
$l_2 \rightarrow \cdots$	0	1	1	1	0	0	0	0 ⋯
$l_3 \rightarrow \cdots$	0	0	0	1	1	1	1	0 ⋯

我们现在知道对列 2、5、7、8 的正确置换是 5、2、7、8。如果考虑更多的行,则可确定更多的列。但如何处理额外的行?如前所见,在放置第三行时我们没有选择,对同一组分的所有其他行亦是如此。对于新行 k,我们所能做的是找出两个已放置好的行 i 和 j,使得图 G_C 中存在边 (k,i) 和 (i,j),并像前面处理第三行那样处理。注意算法如何在上述每个列的集合中保留了组分的可能置换。注意到在放置首两行之后对其余行没有别的放置选择,我们发现这个算法给出了含 C1P 的组分的所有可能的列置换,包括置换的反转。在上面的例子中,如果组分仅有 3 行,两种可能的置换是 5、2、7、8、1、4 和 5、2、7、8、4、1。如我们先前所提及,保留所有可能解的轨迹是 DNA 标图的一个好特征。

上述算法有一个简单的实现:构建图 G_C 并进行深度优先搜索,每当遇到结点时调用过程 *Place*,见图 5.8。为检查算法是否找到一个 C1P 置换,我们可以在每当尝试一个新放置时检验列集合的一致性,如过程 *Place* 最后一行所示。例如,如果我们试图放置一个新行并发现它的一列应该在右边,而现有的放置告诉我们它应该在左边,则这个组分以及这个矩阵不含 C1P。

Algorithm *Place*
  input: u, v, w, vertices of $G_C = (V, E)$ such that $(u, v) \in E$ and $(v, w) \in E$.
  // *v and w can have value nil*
  output: A placement for row u, if possible
  if $v = nil$ **and** $w = nil$ **then**
    Place all 1s of row u consecutively
  else if $w = nil$ **then**
    Left- or right-place the 1s of u with respect to the 1s of v
    Record direction used
  else
    if $u \cdot w < \min(u \cdot v, v \cdot w)$ **then**
      Place u with respect to v using same direction used in v, w placement
      Record direction used
    else
      Place u with respect to v using opposite direction used in v, w placement
      Record direction used
  Check consistency of column sets.

图 5.8　给定两行时,放置一行的过程　157

我们分析一下这个算法的运行时间。构建图 G_C 耗费 $O(nm)$ 时间。然后,如前面所述处理 n 个行,每一行耗费 $O(m)$ 时间以检测列集合的一致性,因此总的时间是 $O(nm)$。

= = = = = = = = = = = 融合组分 = = = = = = = = = = =

我们现在处理算法的第二部分,即组分融合(component-joining),为此需要另一个图 G_M,它将告诉我们组分之间如何相互适配。原始矩阵 M 的每一个组分将是 G_M 的一个顶点。如果对所有的 $i \in \beta$,相应的集合 S_i 都包含在组分 α 的至少一个集合 S_j 中,则顶点 α 和顶点 β 之间存在一个有向边(此处我们不区别行 j 和其对应的集合 S_j)。对应于表 5.1 的矩阵,其组分的图 G_M 见图 5.9 所示。

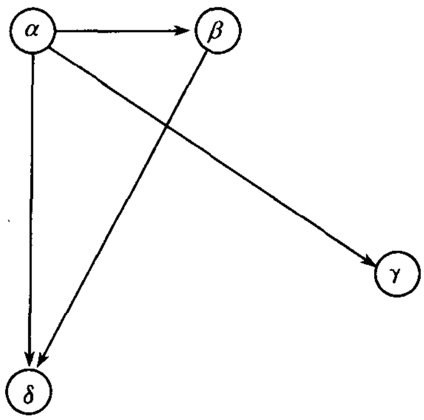


图 5.9 对应于表 5.1 的矩阵的组分的图 G_M

由本节开始的讨论，我们知道 β 的一个集合和 α 的一组集合之间的包含关系与 β 的所有集合之间的包含关系是一样的。让我们阐述的更清楚些。假定属于组分 β 的集合 S_i 被包含于属于另一个组分 α 的集合 S_j，因为行 i 和 j 属于不同的组分，在组分 α 中没有行 k 使得 S_i 不被包含于 S_k 中且 $S_i \cap S_k \neq \phi$，换言之，S_i 被包含于 α 的某些集合中且与其他集合分离。我们可以说，同样的包含和分离对 β 的所有其他集合都成立。倘若不是，让我们分析一个这个异常的情况。假定 $i, l \in \beta, S_i \cap S_l \neq \phi, j \in \alpha, S_i \subseteq S_j, S_l \not\subset S_j$，在这种情况下，$l$ 不属于 α 的惟一可能是 $S_l \cap S_j = \phi$，但这不可能，因为 S_i 和 S_j 至少有一个相同元素，且 S_i 包含于 S_j 中。我们把其他的情况留给读者。

组分的融合依赖于一个组分的集合与另一个组分的集合的包含与被包含关系。直觉
158 上，我们应该先处理有集合不被任何其他集合包含的第一个组分。因为包含由边的方向给定，我们不得不按拓扑次序处理 G_M(如果 G_M 是无环的，我们仅能这样处理，难道不是吗？这不难证明)。

正确地置换所有组分的列之后，我们选择 G_M 中所有没有入边的顶点，并固定它们的列。然后，按拓扑次序处理下一个顶点。假定我们正处理边(α,β)，这意味着我们必须在组分 α 中找出某种“参照列”，这些参照列可以告诉我们如何放置 β 的各行。从 β 中选择具有最左端 1 的行 l，把有 1 的这列称做 c_β。我们知道 S_l 包含于 α 的某个 S_i 中，但不在其他集合中。在 α 中找出所有包含 S_l 的行，并找出满足所有行均为 1 的最左一列(称这列为 c_α)，这就是我们要找的参照列，因为我们现在能使 c_α 和 c_β 相同。

让我们用表 5.1 的矩阵和图 5.9 来展示这个过程，并按拓扑次序考虑各个组分。其中的一个排序是 α、β、δ 和 γ。这里是组分 α 的各行，其列已置换，使得 1 连续：

	{1}	{2,4,5,7,9}					{3,6,8}			
$l_1 \to \cdots$	1	1	1	1	1	1	0	0	0	…
$l_2 \to \cdots$	0	1	1	1	1	1	1	1	1	…

现在融合下一个组分 β，它是单独的一行：

		{2,4,5,7,9}					
$l_3 \rightarrow$	…	1	1	1	1	1	…

我们将看到，所需做的是融合 l_3 与 α 中的行，无需任何重排：

		{1}	{2,4,5,7,9}					{3,6,8}			
$l_1 \rightarrow$	…	1	1	1	1	1	1	0	0	0	…
$l_2 \rightarrow$	…	0	1	1	1	1	1	1	1	1	…
$l_3 \rightarrow$	…	0	1	1	1	1	1	0	0	0	…

下一个组分是 δ，其行含连续 1：

	…	{9,5}		{4}	{7}	{2}	…
$l_6 \rightarrow$	…	0	0	1	1	0	…
$l_7 \rightarrow$	…	0	0	0	1	1	…
$l_8 \rightarrow$	…	1	1	1	0	0	…

我们可以看到在组分 δ 中，l_8 具有最左端 1(在列 5 或列 9 中)。在组分 α 中，行 l_1 和行 l_2 含 l_8，而第二列是全 1。因此，第二列应与组分 δ 中的第 9 或第 5 列对应，得到下面的结果： 159

		{1}	{9,5}		{4}	{7}	{2}	{3,6,8}			…
$l_1 \rightarrow$	…	1	1	1	1	1	1	0	0	0	…
$l_2 \rightarrow$	…	0	1	1	1	1	1	1	1	1	…
$l_3 \rightarrow$	…	0	1	1	1	1	1	0	0	0	…
$l_6 \rightarrow$	…	0	0	0	1	1	0	0	0	0	…
$l_7 \rightarrow$	…	0	0	0	0	1	1	0	0	0	…
$l_8 \rightarrow$	…	0	1	1	1	0	0	0	0	0	…

最后我们“黏上”组分 γ，其各行是：

		{6}	{3}	{8}	
$l_4 \rightarrow$	…	0	1	1	…
$l_5 \rightarrow$	…	1	1	0	…

融合上面各行，我们得到最后的带全部的连续 1 的矩阵：

	{1}	{9,5}	{4}	{7}	{2}	{6}	{3}	{8}
$l_1 \rightarrow$	1	1	1	1	1	1	0	0	0
$l_2 \rightarrow$	0	1	1	1	1	1	1	1	1
$l_3 \rightarrow$	0	1	1	1	1	1	0	0	0
$l_6 \rightarrow$	0	0	0	1	1	0	0	0	0
$l_7 \rightarrow$	0	0	0	0	1	1	0	0	0
$l_8 \rightarrow$	0	1	1	1	0	0	0	0	0
$l_4 \rightarrow$	0	0	0	0	0	0	0	1	1
$l_5 \rightarrow$	0	0	0	0	0	0	1	1	0

注意,其中一个列集合不是单元素集合,这提示我们已找到两个解。在这个特殊例子里,列 5 和列 9 恰好是相同的。但一般地我们会得到许多不包含相同列置换的解。例如,1、2、7、4、9、5、8、3、6 是这个矩阵的另一个解。因为 G_M 可有不同的拓扑排序,因此可有多种解存在,况且每个组分自身还可有几个解,其中的每一个都可有两种方式被使用(置换及其反转)。

现在我们分析算法的运行时间。G_M 的拓扑排序需要时间 $O(n+m)$。可以先预处理 M 的入口项,使得遍历 G_M 时的查询操作仅耗费常数时间。例如,对每一行我们可以储存最左侧 1 的列的位置。这种预处理至多耗费 $O(nm)$ 的时间,因此这也是融合组分所需要的时间。

另外一种算法能在 $O(n+m+r)$ 的时间里发现 C1P,这里 r 是矩阵中所有 1 的数量。此外,算法还以紧凑的方式编码所有的解。不过,这个算法也比我们在本节介绍的算
160 法更为复杂。有关的线索在文献提要中给出。

5.4 带错杂交作图的一种近似

在前节中我们研究了解决连续 1 问题的算法。我们看到,当不含错误且探针是惟一的时,这个问题是杂交标图的一个好模型。

让我们首先检查在克隆×探针二进制矩阵 M 中会发生影响的错误。假定 M 带有真的列置换,给定一行,如果它没有错误,则所有的 1 是连续的。如果一行对应于一个嵌合克隆,其中有两个片段是融合的,我们将会看到连续 1 被一些 0 分成两块(假定该行中无其他错误),并称这种两侧被连续 1 块所夹的连续 0 块为**缺口**(gap)。注意此处缺口一词的用法与本书其他章节中的缺口不同,我们因此可以说在这一行中缺口是因嵌合片段而产生的。如果在另外一行有一个假阴性,则对应的 0 将连续 1 分为两块,产生如下所示的另外一个缺口:

$$0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0$$

↑（指向第 4 位的 0）

一个假阴性

如果探针位于最左侧或最右侧,则不会产生缺口。最后,假阳性也会把一个连续 0 块

分成两个,产生另一个缺口。因此我们看到在矩阵中错误和缺口之间存在着紧密的对应。假定我们力求避免以实验错误解释缺口,一个合理的做法是试图发现一个矩阵中缺口总数最小的置换。这样的做法具有良好的性质,即如果存在一个 C1P 置换,它将具有最少的缺口数。换言之,缺口最少化可以看做是连续 1 问题的一般化。在 5.2.3 已述及 C1P 问题的某些扩展是 NP-难题,缺口最小化同样如此。但对这个特定的 NP-难题,我们可以应用许多特殊的技术获得近似解,且我们可以期望这样的近似解相当好,下面将会看到这一点。

5.4.1　一个图模型

我们已知缺口最少化等价于求解一个熟知的图论难题,即 2.2 节中描述的旅行售货员问题(TSP)。

对于这种 TSP 问题,输入是完全无向加权图 G,G 的顶点对应于克隆×探针二进制矩阵 M 的列,即对应于探针。出于即将明了的理由,我们还必须给 M 增加一个全为 0 的列,且 G 也须有这对应的顶点。G 的每一条边的权是两对应列不同的行的数量(也叫做 161
行之间的汉明距离,Hamming distance)。例如,在表 5.3 中我们有一个二进制矩阵的例子,而图 5.10 则是对应的图。我们现在证明图 G 的最小权环对应于 M 中的一个缺口数最少的列置换。

表 5.3　带附加列 p_6^* 的克隆×探针矩阵

	p_1	p_2	p_3	p_4	p_5	p_6^*
c_1	1	1	1	0	0	0
c_2	0	1	1	1	0	0
c_3	1	0	0	1	1	0
c_4	1	1	1	1	0	0

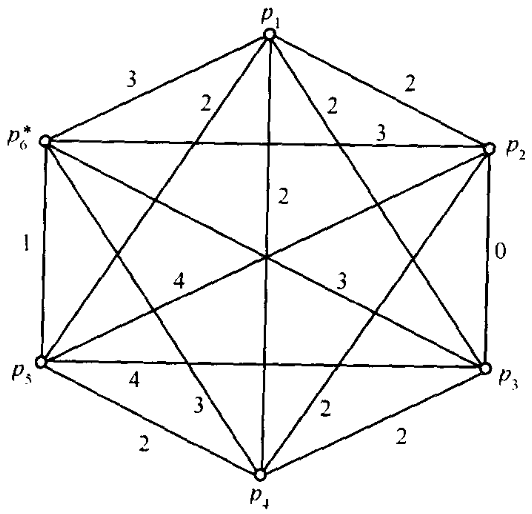


图 5.10　表 5.3 矩阵的 TSP 图

为证实这一点,注意对给定的列置换,行中的缺口意味着在某个点有一个从 1 到 0 的

转换，而且还将有一个从 0 到 1 的转换。所以，每一个缺口对应于两个转换，而对于对应给定列置换的环的权，每一个缺口的贡献恰好是 2。可是，**极值转换**(extremal transition)，即发生在极值列(1 或 m)之间的转换，也增加边的权，而它们并不对应缺口。为保证每一行有一对极值转换，我们增加额外的全 0 列置于列 $m+1$。如果没有这一列，图中的环所对应的置换就会发生连续 1 在每行缠绕，我们不能让这种情况发生。环和置换的关系现
162 在变为：

$$\text{环的权值} = \text{缺口转换数} + 2n$$

这意味着对于给定的 n，最小化环的权值等同于最小化缺口的数量。

刚才我们所做的是显示如何把缺口数量最小化问题简约为 TSP 问题。众所周知，TSP 是 NP-难题，所以原则上我们没有取得太多的进展。但是，已有一大堆求解或近似求解 TSP 问题的技术，这些技术现在就可以为我们所用。这些技术本身并不足以使我们确信解决 TSP 问题能够给出一个真正的探针置换。我们需要的是这样一个保证，即在某种意义上这样的解确实接近真实的解。我们将在 5.4.2 显示一个这样的保证。

在此之前，让我们重新回到缺口最小化问题，并介绍一个体现 5.2.4 末尾所提出的想法的例子。对于给定的输入矩阵，我们已定义了一个函数返回矩阵中缺口的总数。我们将进一步证实，获得一个具有该函数的最小函数值(或接近最小值)的置换将有助于我们找到真实的列置换。由于我们不能保证真实的置换存在于所发现的解中，我们需要寻找其他可能有用的函数。其想法是，通过仔细定义几个这样的优化函数并发展出相应算法，我们将能增加找到真实解的可能性。特别地，可以预期真实解存在于所有解集合的交集中。不过，如果每个函数表征一种真实解确实有或很可能有的特性，则情形将恰如此。

这是另一个优化函数的一个例子。缺口最小化的一个可能缺陷是，在一个相关函数值最小的置换中，一行或多个行可能会有许多缺口，而其他行却完全没有缺口。一行中有许多缺口是不好的，这意味着某个克隆的错误要比其他克隆多许多，这与实验室情况相矛盾。所以我们尽量将每一行的缺口数量最小化。在这种情况下如何应用先前的图模型是习题 15 的问题，其产生的图模型叫做**瓶颈 TSP 问题**(bottleneck traveling salesman problem)，它同样是 NP-难题。

5.4.2 一个保证

这里我们介绍一个形式证明，证明上段所述的 TSP 方法可以以很高的概率给出一个真实的置换。证明依赖于两个基本的假设：探针的数量足够大，且标图过程遵循某种数学模型，这种模型看来是大规模标图中所发生事件的良好表示。下面描述这个模型。

首先我们假定所处理的 DNA 分子足够长，可看做是实线上的一段区间，从 0 延伸到 N。克隆是这个长区间的子区间，并假定所有的克隆具有同样的长度。为方便起见，每个
163 克隆的长度是一个长度单位。为简化叙述，我们以“克隆置换”代替“探针置换”。单位长度假设使得二者是等价的，这意味着我们将在列中寻找连续 1，而不是在行中寻找，且 TSP 中的每一个顶点对应一个克隆。我们以每一个克隆的左末端作为克隆的位置标记。当然，我们并不知道沿着分子的每一个克隆的精确位置。由于我们处理的是杂交，我们所需要确定的只是克隆的相对次序。

这个模型的一个关键特征是沿目标 DNA 的克隆分布。我们将假定每一个克隆的位置是一个独立随机变量,克隆的位置标记平均分布于$[0,N-1]$上,且克隆覆盖$[0,N]$(也就是说,对于$[0,N]$的每一个子区间 I,总存在至少一个克隆 C 使得$C\cap I\neq\phi$)。

这个模型的另外一个重要方面是探针的分布。我们将不假设探针是惟一的,但假定探针沿着目标 DNA 分子稀疏地分布。更形式化地,我们说给定探针的出现遵循泊松过程,而且,任何一个探针的泊松过程是独立于其他探针的。模型的这一部分立刻让我们得到特定探针与一个克隆杂交的概率表述,表达式为:

$$\Pr\{\text{在给定克隆中给定探针出现 } k \text{ 次}\} = e^{-\lambda}\frac{\lambda^k}{k!} \tag{5.1}$$

根据我们的克隆是单位长度区间这一事实,这个表达式可从任何一本有关泊松过程的教科书中得到。

这样完成了模型的描述。我们将要详细说明,给定了克隆×探针二进制矩阵,解决相关 TSP 问题的行置换是真实置换的一个良好的近似。随着探针数量的增加,两种置换的概率均趋近于 1。注意,对一个给定的问题实例,探针的数量是固定的。我们断言,在越来越大的问题实例里,TSP 置换亦以越来越大的概率与真实置换趋同。

为了证明这个论断,我们必须使用图的权(或更合适地,克隆距离)。前已述及,与输入矩阵 M 相关的图的每一条边的权被称作两个末端(克隆)之间的汉明距离。克隆 i 和 j 之间的汉明距离记做 h_{ij}(我们也可考虑到克隆之间的真实距离)。用 l(左)和 r(右)表征一个克隆的坐标,可以把这个距离定义为:

$$t_{ij} = |l_j - l_i| + |r_j - r_i| = 2|l_j - l_i| \tag{5.2}$$

假定克隆都具有同样的大小。

假设现在我们已知道了全部真实距离,那么很明显最大的距离将告诉我们距离最远的一对克隆,也就是说位于区间$[0,N]$顶端的克隆;次最远的距离则相似性地给出另一对克隆,出现在前面一对之内;依此类推。也就是说,给定真实的距离我们可以得到真实的克隆置换,这并不是一个令人惊奇的结论。不过,我们仅仅知道距离 h_{ij} 而不是真实的距离 t_{ij}。由于我们仅仅想得到克隆的相对次序,这样的信息便是充分的。例如,如果我们说对任何克隆 i、j、r、s,$h_{ij}<h_{rs}$,则 $t_{ij}<t_{rs}$,反之亦然。原因是,为了用真实的距离依序放 164
置克隆,我们所需知道的仅是一些有关克隆距离之间次序的概念。如果我们能够证明,$h_{ij}<h_{rs}\Leftrightarrow t_{ij}<t_{rs}$的概率随探针数量的增加而趋向于 1,则我们就可获得所需要的结果。

让我们考察克隆对 i 和 j,并找出某个探针 p 对汉明距离产生贡献的概率。如果探针 p 与 i 杂交而不与 j 杂交,或情况相反,它就对汉明距离产生贡献。根据式 5.1,可以看到探针不出现在克隆 j 上的概率为 $e^{-\lambda}$,因为 $k=0$。另一方面,探针 p 至少出现在克隆 i 上一次的概率与探针不出现在与 j 不重叠的 i 的节段上的概率的补是相同的。如果这个重叠为 z_{ij},则我们得到下面的结果:

$$\Pr\{p \text{ 与 } i \text{ 杂交而不与 } j \text{ 杂交,或相反}\} = p_{ij} = 2e^{-\lambda}(1-e^{-\lambda(1-z_{ij})}) \tag{5.3}$$

这本质上意味着对这个事件的发生存在着一个明确定义的概率。现在我们考虑所有的探针,每一个都具有与克隆 i 和 j 杂交的概率。假设有 m 个探针,考虑 h_{ij}/m,它表示每一个探针对克隆 i 和 j 之间汉明距离的平均贡献(注意探针对汉明距离的贡献是 1 或

0)。如果探针的数量很大，我们可以援用大数定律，称 h_{ij}/m 接近于 p_{ij}。或者，对任意固定的小的正实数 ε，当时 $m\to\infty$ 时，

$$\Pr\left\{\left|\frac{h_{ij}}{m}-p_{ij}\right|>\varepsilon\right\}\to 0$$

注意，由于克隆是单位长度，两个交叠克隆 i 和 j 之间的真实距离为 $t_{ij}=2-2z_{ij}$，交叠由 z_{ij} 表示(见等式 5.2)，这使得我们能以 t_{ij} 代替式 5.3 中的 z_{ij}，这显然表明对任意克隆对 i、j 和 r、s，当且仅当 $p_{ij}>p_{rs}$ 时，$t_{ij}>t_{rs}$。而如果情况如此，我们则可以说：

$$\Pr\{|t_{ij}-h_{ij}|>\varepsilon\}\to 0$$

这意味着 $h_{ij}<h_{rs}\Leftrightarrow t_{ij}<t_{rs}$，这正是我们要证明的。

5.4.3 实际计算

本节介绍应用杂交标图算法进行计算机测试的一些实际结果。在介绍这些结果之前，我们必须考虑这些测试结果的解释方式，其中的一些考虑亦适用于计算生物学中的其他许多问题。

开始我们将用另一个图来检查输入数据，通过这种方式我们可以清楚地看到期望什么样的算法来试图解决真实的探针置换。下一部分也要用到这个图。

我们把杂交图 H(hybridization graph H)定义为二部图(bipartite graph)(U,V,E)，它用杂交矩阵的信息构建，克隆是 U 分区的顶点，探针是 V 分区的顶点。如果探针与对
165 应克隆杂交，则在两个顶点间存在一条边。在图 5.11 中我们可以看到基于表 5.3 的矩阵 M 构建的二部图 H，它排除了全 0 的列。

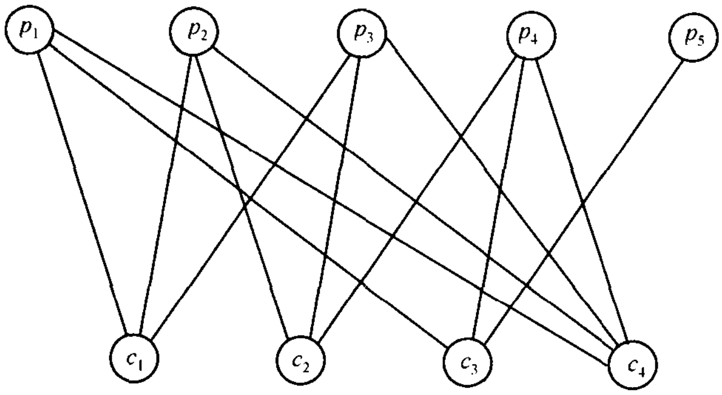


图 5.11　对应于表 5.3 的杂交矩阵的杂交图 H，无增加的列

首先要注意的是，即便杂交矩阵中的所有入口项都正确，H 也可以是非连通的。如果是这种情况，不管我们的算法如何好，我们将无从知道属于不同连通组分的探针间的相对次序，因为所需信息并不存在于这样的矩阵中。一个连通组分可能简单到是一个孤立顶点，意味着有一个不与任何克隆杂交的探针或者有一个不与任何探针杂交的克隆。另外一个现象是可以存在冗余探针，或与同一克隆集合杂交的探针。如果探针(尽管是不同的)杂交到目标 DNA 上紧密相关的部分，这种情况就会发生。如果特定 DNA 片段的某些克隆丢失了，使得一些探针无任何阳性杂交结果，也会发生这种情况。

当我们解决相应的连续 1 问题时，H 的连通组分便显现出来，冗余探针也可很容易地在杂交矩阵中看到，它恰巧是具有同样 1 和 0 的列。但如果存在错误，我们就会得到 H 的连通组分的数量和结构以及一个探针对是否冗余的错误信息。我们说错误可能"掩盖"了有关的信息。因此，必须注意引发错误的这种困难情况：不含错误的输入矩阵可能缺乏发现真实置换所必需的信息（例如，无错的 H 含有几个组分），而发现这些缺乏的信息十分困难（因为无错 H 仅有一个组分）。假定表 5.3 中的矩阵是无错的，探针 p_2 和 p_3 是冗余的，但如果克隆 c_3 和探针 p_3 之间存在一个假阳性，我们将无法发现它。

对于这种情况，很明显评估一个作图算法是一件很困难的事（更何况还有作图本身!）。假定我们已多少知道作图问题的正确答案，现在看一看如何评估这样的算法。如果我们用计算机程序产生作图问题的人工例子，并模拟实验错误，这种做法就是能行的。如果这样的例子与真实情况无异，通过上面的讨论将会很清楚地看到输入矩阵可能缺乏 166
能够使算法确定真实探针次序的信息。因此我们必须试着评估由特定作图算法发现的解如何"接近"于探针的真实次序，此时问题变成了：我们如何定义"接近性"？目前对此尚无一致意见。尽管如此，为了对当前作图算法的性能做些评估，作为例子我们将介绍文献中出现的一个定义。这是一个合理的定义，但即便广为接受，它仍需作某些精炼。

我们将利用作图算法所报出的"*强邻接*(strong adjacency)"的多少来度量一个作图算法。强邻接是通过杂交矩阵中出现的连续 1 的块数 b 和给定的探针置换 $\pi = p_1, p_2, \cdots, p_m$ 来定义的。我们还分析*转位*(translocation)效应，它是将一组连续探针的次序反转的操作。对两个邻接探针 p_i 和 p_{i+1}，如果在每行中用转位将这些探针分开就导致增加连续 1 的块数 b，则这两个探针就代表一个强邻接。当这种 b 值的增加发生时，我们会有若干证据表明探针 p_i 和 p_{i+1} 在所有的解中都应该是邻接的。

基于这个概念，我们能够定义一个给定置换的*强邻接开支*(strong adjacency cost)，它由下面的公式描述

$$100\left(\frac{1}{m-1}\sum_{0}^{m-2}\delta_i\right) \tag{5.4}$$

这里，如果 p_i 和 p_{i+1} 在一个真正的置换中是一个强邻接，但在给出的置换中却不是邻接的，则 $\delta_i = 1$；否则 $\delta_i = 0$。注意，开支是一个百分数，好的置换必须有低的强邻接开支。

使用这个定义，我们最终能够给读者一个有关算法性能的概念。表 5.4 刻画了两个算法的强邻接开支。其中一个是"随机"的，即探针置换是随机的；另一个基于 TSP 方法，下面描述它如何工作。给定一个 TSP 图，通过选择顶点对和使它们在一个通路上邻接用算法来构造一个环。在每一次迭代中选择的顶点对必须满足下列条件：如果 u 和 v 已经属于通路，且这些通路是不同的，则仅当 u 和 v 是它们各自通路的端点时这些通路可以被融合；其次，在所有的合格顶点对中它们必须是最紧邻的。当所有的顶点都在同一通路上时，该通路闭合而构成一个环。这个解随后被用于另一个算法，后者试图用另一个启发式来改进解。

在表 5.4 中，我们可以看到相比于"随机"算法，基于 TSP 的算法（称做"贪婪"算法）性能更佳。给出这些结果的论文介绍了另外 3 个更成熟的算法的性能，其结果与上述"贪婪"TSP 算法相似。这可看做支持 TSP 方法的一个证据，但另一方面它也说明标图问题

167 的困难性。该表也显示出现假阴性时贪婪 TSP 算法的解非常差。因此可以假定，当所有类型的错误均在同一例子中出现时，结果将更糟。这促使我们研究下部分的议题，即在假阴性出现时使用一个相当强健的启发式。

表 5.4　两个算法的在矩阵上的强邻接开支，带有不同类型的错误。错误率由每一列的标题给出。所有例子中的覆盖是 10，这里覆盖是所有克隆总长度和目标 DNA 长度之间的比率

	C1P	嵌合性	假阳性	假阴性
	0	0.5	0.04	0.32
贪婪 TSP 算法	1.9	0.9	16.0	28.3
随机算法	86.4	89.7	94.4	94.9

5.5　杂交标图的启发式

同前节所示，标图是一个非常困难的问题，尚未发现一般化的和好的算法。其后果是，我们看到在实践中研究者求助于各种启发式以帮助他们得到解。本节给出两种启发式，它们在杂交标图中可以产生好的结果。

5.5.1　筛选嵌合克隆

前面已注意到，嵌合克隆在克隆文库中以较高的频率发生，它们的出现给任何标图算法都带来严重的问题。这一部分给出一个简单的启发式，试图将嵌合克隆拆成片段。这样的启发式用做*筛选过程*(screening procedure)非常有用，可作为应用复杂技术之前的预处理。

概念非常简单：如果一个克隆是嵌合的(假定它仅由两个片段组成)，与其中一个片段杂交的探针不应该同与另一个片段杂交的探针有关联。这里的关键点当然是“关联性

168 (relatedness)”的概念。通过图 5.12 可以弄得更清楚。

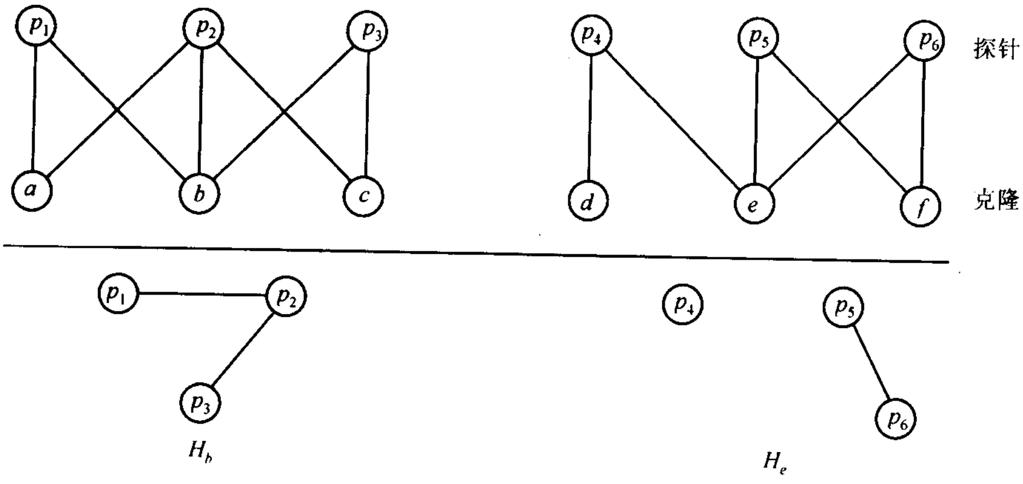


图 5.12　横线之上是两个克隆×探针矩阵的杂交图 H。线之下左侧是克隆 b 的图 H_b，这个克隆可能不是嵌合的，因为 H_b 连通。右侧是对应克隆 e 的图 H_e，这个克隆可能是嵌合的，因为 H_e 不连通

取克隆 i 及与其杂交的探针集合 P_i。对每一个克隆 i,我们创建图 $H_i=(P_i,E)$。如果 P_i 的两个探针与非 i 克隆杂交,则在两个探针之间创建边。如果所获得的图是连通的,我们说 i 不嵌合;如果图的组分超过一个,我们就说 i 是嵌合的,并以一个新的"人工"克隆代替 i,每一个新克隆由 H_i 的一个连通组分给定。这个方法可进一步改进,要求的条件是,仅当探针 p 和 q 与至少 k 个其他克隆杂交时,H_i 中 p 和 q 之间才存在边,这里 k 依赖于手头要解决的问题的参数。

经验表明这种启发运行良好。但另一方面,它有可能会把一个实际上非嵌合的克隆认为是嵌合的。因此,另一个有用的启发式是将原本是一个克隆的两个克隆结合起来。

5.5.2　获得一个好的探针次序

这里呈示的启发式比上部分介绍的功能更强,其目标是实际地解决这个问题,即获得一个探针的置换。基本思想是通过查看杂交图 H(在 5.4.3 定义),对每个探针 p 估计其左侧探针的数量与右侧探针的数量。

给定一个探针,如果我们能够以某种方式将其他探针分成两个不同的组分:一个左侧组分和一个右侧组分,那么就能记数该探针左侧与右侧探针的个数。鉴于靠近目标 DNA 末端的探针无法将其他探针分成两部分,因此我们把能够获得两个组分的探针称为分裂元(splitter)。检测分裂元的方法将在下面介绍。

给定探针 p,可构造一个顶点集合 S_p(其元素是克隆和探针),办法是把 p 以及与 p 共享一个克隆的所有探针置于 S_p,再把与 S_p 中任何探针相关的克隆包含入 S_p。现在,从 H 中移走 S_p 中的所有顶点。如果产生的图恰巧有两个组分,我们就说 p 是一个分裂元。应用这种方法,可得到一个图 5.13 描述的启发式算法,它能获得一个好的探针次序。 169

Algorithm *Probe Permutation Heuristic*
　input: clones × probes hybridization matrix
　output: a "good" permutation for the probes
　Identify splitters as described in the text
　for every splitter i **do** Determine the components A_i and B_i
　for each probe p **do** Initialize l_p and r_p with zero
　Select a pair (A_k, B_k) arbitrarily
　for each probe p **do**
　　for each pair of components (A_i, B_i) **do**
　　　if $p \in A_i$ then
　　　　if $|A_i \cap A_k| > |A_i \cap B_k|$ **then**
　　　　　Increment l_p
　　　　else
　　　　　Increment r_p
　　　else if $p \in B_i$ **then**
　　　　if $|B_i \cap A_k| > |B_i \cap B_k|$ **then**
　　　　　Increment l_p
　　　　else
　　　　　Increment r_p
　Sort probes in decreasing order of $l_p - r_p$.

图 5.13　获得好的探针置换的启发式算法

对每一个探针 p,算法保留了两个记数器,l_p 和 r_p,分别记录含 p 的左侧和右侧的区域数量。这些记数器初始值为 0,应用分裂元时记数增加。如何做到这一点尚不清楚,因为尽管我们知道一个分裂元 i 有两个组分 A_i 和 B_i,但我们并不知道哪一个在哪一侧。为解决这个困难,我们需要一个固定的任意分裂元 k 来作为参照。我们假定 A_k 在 B_k 的左侧,然后给定另外一个分裂器 i,当 $|X\cap A_k|>|X\cap B_k|$ 时,i 的一个组分 X(或是 A_k 或是 B_k)是最左侧的一个。一旦我们获得最终的记数,我们就排序探针,使得具有较高左偏差值 l_p-r_p 的探针列于前面。对这个启发式算法的一个简单改进在于分裂元 k 的选择,这个分裂元越"中间",效果就越好。

如同前面所述的启发式,经验表明这个启发式算法运行得很好。特别是,对于假阴性错误,它表现得相当稳定。此外,这个启发式亦可为其他算法或启发式算法提供计算结果,以便提高算法性能。

小　　结

本章我们研究了用于为 DNA 作图而产生数据的两种方法:限制酶消化和杂交实验。
170 应用限制酶数据的一种方式是测定和比较对应片段的长度,这可导致双重消化问题和部分消化问题。我们说明了双重消化问题是 NP-完全的。有关片段的数据还可用于确定片段的指纹,在这种情况下,通过按片段指纹确定片段间的交叠可重构片段的次序,这可通过各种方法进行模拟,使用区间图为工具,但大多数模型均导致 NP-完全问题。

在杂交标图中,克隆×探针杂交矩阵是原始数据。如果对这个矩阵赋予严格的要求,即探针是惟一的且没有错误,那么所对应的问题可在多项式时间求解。我们所需要做的是确定这个矩阵是否有连续 1 特性,我们给出了针对这个问题的一个算法。当原始数据存在错误时,则必须求助于近似方法,且我们已表明标图问题可简约成 TSP 的一种形式。结合杂交过程的另一个模型,这种简约可产生一个保证,即对 TSP 的一个解将非常接近标图问题的解。标图算法的困难性促使我们寻找算法评估的标准。我们呈示了其中的一个标准,表明在这种标准下,相比于其他更复杂的途径,基于 TSP 的途径表现良好。最后,我们介绍了两个启发式算法,用于筛查错误和发现好的探针次序。

习　　题

1. 找出一个图 5.2 中问题的候选解。
2. 给定下列双重消化实验结果,试图发现一个解。

   酶 A: 4,5,7,8,12

   酶 B: 3,4,4,6,9,10

   酶 $A+B$: 1,2,3,3,4,4,4,4,5,6
3. 给定下列部分消化实验结果,试图发现一个解: 2,3,7,8,9,10,11,12,17,18,19,21,26,29。
4. 设计一个求解 DDP 问题的穷尽搜索算法。

5. 解释为什么两个不同限制酶的限制位点不会重合。

*6. 图 G 是一个区间图的一个必要条件是 G 中的任何环,如果包含 4 个或更多个顶点,都有一个弦。也就是说,一个连接两个不连续顶点的边属于环。证明之。

7. 证明习题 6 中的条件不是充分的。

8. 对一个区间图,如果每一区间都不被完全包含在任意其他区间中,则该区间图等价于一个所有区间都等长的区间图。证明之。

9. 执行 5.3 节的 C1P 算法,输入矩阵由表 5.5 给出,且有一个列置换使得每一行都有连续的 1。

10. 刻划每一行以及每一列都有 C1P 的二元矩阵。 171

* *11. 使用 C1P 刻划区间图,即给定一个图 G,提出一个基于 G 的二元矩阵 M,使得 G 是一个区间图当且仅当 M 有 C1P。

*12. 基于习题 11 的答案,设计一个有效的算法识别区间图。

13. 在 5.3 节的 C1P 算法中,证明:组分 β 的一个集合与另一个组分 α 的所有集合之间的包含关系与 β 的所有集合间的包含关系是一样的。这是一个个例的证明,正文中已给出了一例。

14. 证明 5.3 节中定义的图 G_M 是无环的。

15. 提出一个模型,用于每行空隙的最小化。5.4.1 的末尾给出了暗示。

16. 证明:在一个二元矩阵中获得一个列置换使得全部连续 1 的块的数量最小,该问题的判定版本是一个 NP-完全问题。提示:由哈密顿通路问题进行简约。

17. 对图 5.10 运行 5.4.3 末尾刻划的 TSP 贪婪算法。

18. 理论与实际结果显示,如果距离遵循 3.6.1 中给出的约束,则求解 TSP 问题可略为容易。汉明距离是否也遵循那些约束?

◇19. 杂交实验有时并不出结果,因为探针与克隆可以发生或不发生杂交。假定每个实验的结果是一个介于 0 与 1 之间的实数,该数给出实际发生杂交的可信度。提出一个算法找出基于这种数据的真正的探针置换。

*20. 根据求解 TSP 问题的最小生成树启发式,提出一个杂交标图算法。关于这种启发式的描述可在文献[152]中找到。

21. 在杂交标图中,有时我们知道在所有探针中有某一些与克隆杂交,并具体知道是哪一些探针。这样的信息可以产生什么样的矩阵?如何重新描述问题使其可被求解?它是否比不带末端探针信息的版本更易求解?

22. 提出一个启发式用于组合被 5.5.1 中描述的筛选启发式错误分隔的克隆。该启发式必须作为后处理步骤,即它必须接收一个由先前步骤获得的探针序列作为输入。 172

23. 在 5.5.2 中描述的一个获得好的探针序列的启发式中,说明一个坏的中间分裂元将会把一个分裂元置于序列中错误的位置。

24. 片段组装和杂交标图是类似的问题。第 4 章中有什么技术能用于杂交标图吗?

25. 研究人类基因组物理作图的主要研究所之一是 MIT 的 Whitehead 生物医学研究所(Whitehead Institute for Biomedical Research)。查看他们的网页 http://www-genome.wi.mit.edu,并查找人类基因组项目的当前进展。

文献提要

关于标图问题的一篇好述评是 Karp 的文章[111]。Lander 给出了一篇关于限制位点标图的出色综述[119],在该文里 Lander 描述了遗传连锁标图(genetic linkage mapping)的算法。遗传连锁标图本文并未涉及,它使用遗传标记作为输入数据,比如限制片段长度多态性(RFLP)。

Goldstein 和 Waterman[73]以及 Schmitt 和 Waterman[169]研究了双重消化标图。有关文献提供了 DDP 是 NP-完全问题的证明[73]。Schmitt 和 Waterman 证明 DDP 解的数量随目标 DNA 的长度呈指数增长[169]。作者还希望存在基于片段等价类的一般解法,但不幸的是 Pevzner 的研究证明这种办法不能实现[160],且得出结论应使用多重消化的途径。Skiena 和 Sundaram 研究了部分消化问题,提出了分支界限算法,并给出了有前景的结果。

Waterman 和 Griggs 对下面的限制位点作图问题提出了一个算法[200]。目标 DNA 由两个酶消化,然后获得每一种消化酶所产生片段的交叠信息。目标是基于这种信息发现限制位点的顺序。

一些论文提供了关于物理作图的计算复杂性分析[72,59,75],5.2.2 即基于这些论文,包括 NP-完全性证明。Kaplan、Shamir 和 Tarjan[108]提出了一个下列与物理作图有关的算法:通过增加尽可能少的边把输入图转换为一个真区间图(proper interval graph)。在一个真区间图中,没有一个区间真正包含另一个区间,换言之,所有的区间均有相等的长度。他们提出了一个算法,如果需增加的边的最大数量是一个固定数 k,则其运行时间是线
173 性的。

5.3 节介绍的连续 1 问题的算法来自于 Fulkerson 和 Gross[67],一个更复杂的算法由 Booth 和 Lueker 给出[27],其运行时间为 $O(n+m+r)$,并用一种紧凑的方法编码所有的解(r 是矩阵中 1 的总数)。不过,该算法十分冗长,因此有人试图加以改进,Hsu[97]与 Meidanis 和 Munuera[136]的工作即是改进的例子。

Greenberg 和 Istrail[78,79]以及 Alizadeh 等[9]研究了探针惟一情况下的杂交作图。前者给出了有关该问题许多方面的出色描述,并介绍了几个算法。5.4.3 的结果即取自该文,而 5.2.4 亦参考了该文。许多其他技术,包括 5.5 节中的启发式,来自于 Alizadeh 等人的论文[9]。Grigoriev、Mott 和 Lehrach 描述了嵌合克隆检测的启发式[80]。Alizadeh 等研究了探针不惟一的杂交标图[8],5.4 节主要基于他们的论文。克隆分布的泊松过程模型取自 Lander 和 Waterman 的工作[120],4.1.2 在讨论覆盖的时候也提到了该模型。这个模型也在作图项目中用到,以确定产生必要覆盖的克隆的数量。

Soderlund 和 Burks 描述了用于物理标图的一个图形界面软件包[177],在该文里他们
174 还提到了用于自动产生测试例的软件。

第 6 章　种系发生树

本章主要描述重构种系发生树(phylogenetic tree)问题，这是生物学的基本问题，它在分子生物学中的用途是帮助理解分子的进化关系，例如蛋白质的进化关系。我们主要介绍基于性状和距离这两种基本输入数据的模型和算法。

现代科学表明，生活在地球上的所有生物物种都随年代而缓慢地转化，这个过程称为进化(evolution)。生物学的一个中心问题是解释今天物种的进化历史，特别是有着共同祖先的物种的亲缘关系如何。这通常以构造树来解决，树叶代表当前物种，而树的节点代表假定的祖先。这类树叫做种系发生树，图 6.1 是种系发生树的一个例子。根据此树，在遗传上人类与黑猩猩的关系比与其他灵长类的关系更接近，这意味着他们有着其他灵长类所没有的共同祖先。该树亦表明了长臂猿与合趾猴具有类似的关系。

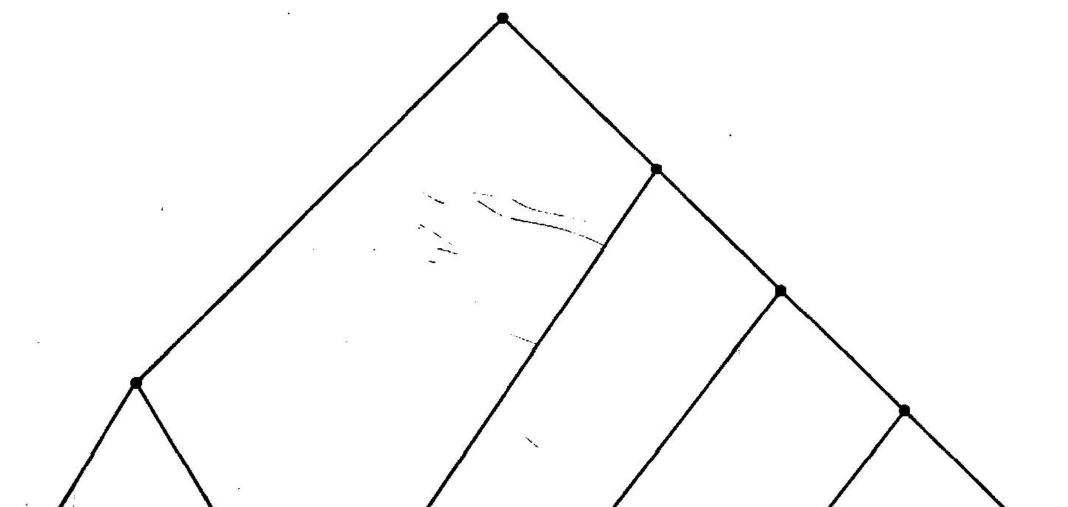


图 6.1　一些灵长类的种系树

构造种系发生树的难题是，我们通常没有关于当前物种远祖的足够资料，即便有，我们也不能 100% 肯定某个化石所属的物种确为两个或多个当前物种的祖先。所以，我们不得不推断当前物种的进化史，并通常用当前物种之间的相互比较作为原始数据来重构

种系发生树。因为这个原因,图 6.1 中的树并不一定是真实的种系发生树,仅是一个假设。

种系发生树,或简单地说种系树,从 19 世纪对物种进行分类时就已提出。直到 20 世纪 50 年代,所应用的方法还是基于研究者的经验和直觉。逐渐地,数学化的形式方法被介绍到种系发生树的研究中。现在已有大量的关于种系发生树重构的数学和算法的工作,还有软件包提供大量程序帮助科学家重构种系发生树。

本章的目标并不是纵览种系发生树重构的计算方法,该论题需要一整本书。我们的
176 目标是研究某些适于算法求解的问题。读者应注意到,鉴于文献的广泛性与多样性,这种处理是非常局限和有偏重的。本章的文献提要给出了更一般性的文献。

在进化生物学中,可以构造物种、群体、属或其他分类学单位(taxonomical unit)的种系发生树(taxonomy 是“分类的科学”,术语数字化分类 numerical taxonomy 用来描述用于任何分类的数字化方法。本章的主题可看做数字化分类的一个子领域)。因为蛋白质和核酸也会进化,所以也可构造它们的种系发生树,这便是分子生物学家为什么对种系发生树的构造方法感兴趣的原因。为了更一般性地说明问题,我们使用术语**对象**(object)来代替用于构造种系树的分类学单位。

回顾一下 2.2 节对树的定义,树是一种无向、非循环和连通的图。我们要区别外部节点(或树叶)与内部节点(内节点)。树叶的度数是 1,而内部节点的度数大于 1。树叶代表研究对象,所以说树叶标记一个或一组对象。

科学家对种系树的两个方面感兴趣,一是其拓扑学结构,即内部节点相互之间以及树叶之间是如何连通的(部分作者以“分枝模式 branching pattern”表示之);另一个是两节点之间的距离,它可在树的边加权时确定,这个距离是对节点间进化距离的一个估计。基于
177 许多我们不打算讨论的假设,内部节点(祖先对象)与树叶(现代对象)间的距离可解释为内部节点物种转化为叶节点物种所经历时间的一个估计。但需注意,进化距离与经历时间通常并不等价。

种系树的另一重要特征是它是否有根。一个有根的树表示内部节点之间有祖先关系。然而,在许多问题中,并没有足够的信息允许我们确定根,在这些例子中重构的树无根。

我们曾经提及种系树是基于对当今物种的比较而构造的。可将构造种系树的输入数据分为两类:

1. 离散性状,如鸟喙的形状,手指的个数,是否出现限制位点等。每个性状有有限的状态。与这些性状相关的数据可以放入对象 × 性状矩阵中,叫做**性状状态矩阵**(character state matrix)。
2. 比较数据,称做物种间的距离,形成的矩阵称为**距离矩阵**(distance matrix)。

每一类数据需要不同的种系树重构方法。这两类数据的最大不同是性状状态矩阵通常不是方阵,因为对象的数量可不等于性状的个数。另一方面,距离矩阵是三角矩阵,因为每一对对象之间均有距离,这些距离是对称的。在后续各节中将介绍这些方法的几种代表性的算法。还有第三类数据称为连续性状数据,但本章未涉及该类数据。

6.1　性状状态和完全种系发生问题

本节首先讨论关于性状的假设与其内部树节点状态的意义。

关于性状的基本假设是很显然的：在种系树重构的过程中该性状被认为是“有意义”的。定义什么是有意义的以及按定义检查被选择的性状是否有意义主要取决于科学家，他们是种系树的“用户”。不过，我们仅简要介绍两个假定，它们是所有基于性状的种系树重构方法的前提。

首先，性状相互间能够独立遗传，这对于我们将讨论的所有算法都至关重要。另一个假设是，对一个给定的性状，所有观察到的状态都是从所研究对象的最近共同祖先的某个“原始状态”进化而来。遵循这个假设的性状称为*同源的*(homologous)。当所研究的对象是生物序列时，这个假设也是重要的，在这种情况下一个性状是序列中的一个位置，若序 178
列是 DNA，其状态则是 A、G、C 和 T。如果我们正在比较位置，比如说序列 s 的 132 位与序列 t 的 722 位，我们必须明了这两个位置的当前状态缘自它们的祖先序列的同一位置。

关于内节点，它们代表的是假定的祖先对象，算法赋予这些节点特定的状态。但一个内节点的一组状态并不一定要具有生物学意义。这意味着内节点的特征化并不用于祖先对象特征的反向预测，主要的目的是对当今对象分组，而祖先对象的构造亦仅是针对此目的。不过，对内节点状态的赋值必须非常小心，以便产生的种系树尽可能地接近真实的种系树。

现在我们转到性状状态矩阵的正式定义。我们把性状状态矩阵定义为有 n 行(对象)m 列(性状)的矩阵 M，M_{ij}表示对象 i 关于性状 j 的状态。每个性状最多有 r 种状态，状态均以非负整数标记，这个矩阵的一个给定行是一个对象的状态向量。由于我们也要给树上的内节点指定状态，因此说内节点亦有状态向量。一个性状状态矩阵的例子见表 6.1。

表 6.1　性状状态矩阵

对　象	性　状				
	c_1	c_2	c_3	c_4	c_5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	1	1	1	0
E	1	1	0	0	1

当试图根据性状状态矩阵构造种系树时，会遇到若干困难。首先是两个或多个对象具有同一性状的相同状态。几乎所有的种系树重构方法都有这样的假定，即共享同样状态的对象比不共享的对象在遗传上更相近。但也存在这样的例子，即两个对象虽共享同一状态但在遗传上却不相近，如蝙蝠和鸟类都有翅膀。这样的现象叫做趋同(convergence)或并行进化。自然界中这样的情况很少见，因此在树重构中处理这种困难的方式是：趋同事件不应发生，或其数量应尽可能少。

第二个困难是如何处理同一性状不同状态之间的关系，我们以表 6.1 的矩阵来说明这个问题。假定有一个祖先对象 X，并由此进化出对象 A 和 B。那么，该如何给 X 指定性状 c_1 的状态？注意，A 中 $c_1=1$，B 中 $c_1=0$。如果使 X 中的 $c_1=0$，则说 0 是祖先状态(ancestral state)，而 1 是衍生状态(derived state)。假定已做出这样的选择，并假定对象 C 179

和 D 有共同的祖先 $Y \neq X$，进一步假定 Y 中 c_1 的状态是 1。在本例中，对象 D 性状 c_1 取状态 0 是对祖先状态的反转。在这个特定的例子中，由于性状是二值的，我们可以用该性状的获得与缺失解释状态值的意义。也就是，A 获得了祖先 X 的该性状，而 D 丢失了祖先 Y 的该性状。如上一部分对困难性的讨论一样，我们观察到在自然界中这种性状状态的反转并不常见。所以，可以用处理趋同事件的方式来处理性状反转，即要求反转不发生或发生数量尽可能少。注意，我们可能并不知道哪一种状态是祖先状态哪一种状态是衍生状态。如果不知道，则必须做出选择，并使所做的选择保持一致，以避免或减少反转。

前面的例子处理的是二值性状。因为通常一个性状可以有 r 种状态，状态之间的关系可能更加复杂。根据对这些关系的了解程度，性状可被分为**有序的**(ordered)和**无序的**(unordered)。对无序性状，我们对状态的变化方式不作任何假定，也就是说状态之间可相互转变。对于有序性状则存在着额外的信息。例如，某一性状的 4 种状态可以有如下的线性次序：$3 \leftrightarrow 1 \leftrightarrow 4 \leftrightarrow 2$，这意味着在重构树中状态 3 不能直接转变为状态 4，需要以状态 1 为过渡。换言之，如果有一个节点，该性状处于状态 3，则不应该有边直接连到该性状为状态 4 的节点。性状亦可有*偏序*(partially ordered)，在这种情况下存在一个*衍生树*(derivation tree)，它显示状态如何通过树互相衍生。注意，对于有序性状我们未曾提及状态发生改变的方向。已经知道这种状态改变的方向的性状称为**有向**的(directed)。在文献中，无序性状亦称做*定性性状*(qualitative character)，有序性状称做*分支性状*(cladistic character)，有向性状则称为*极性性状*(polar character)。直觉上，有序性状能够提供更多的构造种系发生的信息，这种直觉对理解下面一些有关计算复杂性的结果是有用的。

如果想完全避免趋同事件和反转，则根据上面的考虑，期望的树 T 应具有下列性质：对每一个性状 c 的每一个状态 s，性状 c 的状态为 s 的所有节点 u(包括树叶和内节点)的集合必然构成 T 的一个子树(即 T 的一个连通子图)。这意味着，连接到该子树的边 e 必然惟一地与一个从状态 w 到状态 s 的转换相关联。具有这种性质的种系发生称为*完全种系发生*(perfect phylogeny)。现在，我们可以陈述基于性状状态矩阵的种系发生重构的核心问题，该问题称为完全种系发生问题。

论题：完全种系发生

实例：有 n 个对象的集合 O，m 个性状的集合 C，每个性状最多有 r 种状态(n, m, r 都是正整数)。

问题：O 是否有一个完全种系发生树？

对于表 6.1 所示的矩阵，假定 0 是祖先的而 1 是衍生的，这样一个种系发生就不存在，这将在下一节说明。对于表 6.2 所示的矩阵，一个完全种系发生则确实存在，并表示在图 6.2 中，图中标记性状的边表明沿该边有一个从状态 0 到状态 1 的转换，使得此边下面的子树包含所有该性状的状态为1的

180 **表 6.2　另一个性状状态矩阵**

对　象	性　状					
	c_1	c_2	c_3	c_4	c_5	c_6
A	0	0	0	1	1	0
B	1	1	0	0	0	0
C	0	0	0	1	1	1
D	1	0	1	0	0	0
E	0	0	0	1	0	0

对象而不含其他对象。

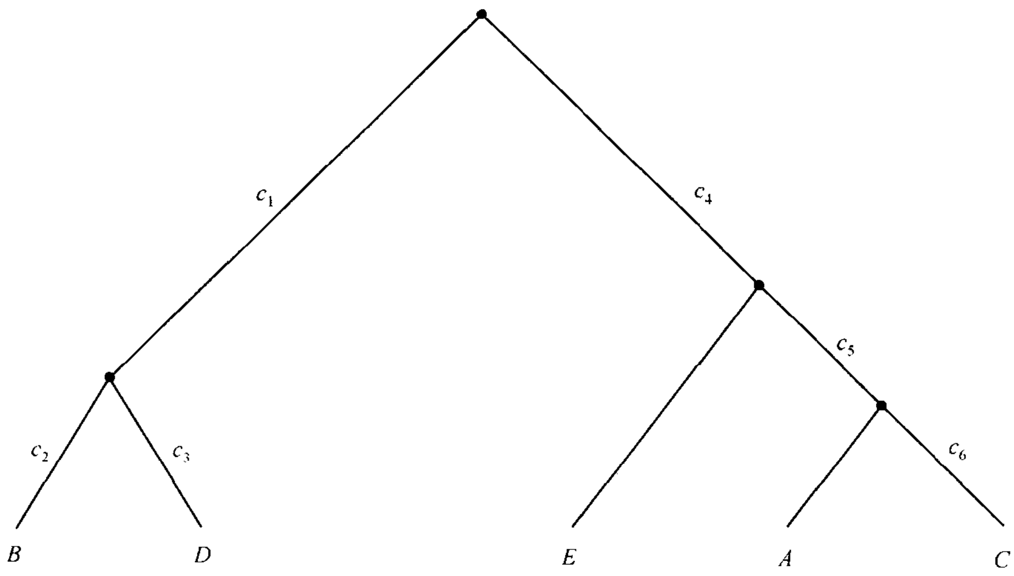


图 6.2　对应于表 6.2 中矩阵的种系发生，发生状态转换的边以对应的性状标记

每当由一个性状状态矩阵所定义的对象集合可产生一个完全种系树时，就说这些性状是相容的(compatible)。这个相容性概念将在下面被研究的特定例子中详细讨论。 181

定义了问题后，要问的第一个问题是：是否存在解决这个问题的有效算法？我们稍后回答这个问题，而首先看看 n 个对象可以构造多少种不同的树。假定我们仅需无根的二叉树，由于对象都在树叶上，那么什么对象在什么部位就不是无关紧要的。对 3 个对象而言只存在一棵树，如图 6.3 所示；对 4 个对象则有 3 棵树，如图 6.4 所示。可以证明(见习题 2)，通常 n 个对象有 $\prod_{i=3}^{n}(2i-5)$ 棵树，这个数字增长得比 $n!$ 更快，因此排除了任何构造所有的树然后从中找出完全种系树的算法。我们希望发现一个解决该问题的算法，其运行时间与 n、m、r 的多项式成比例。

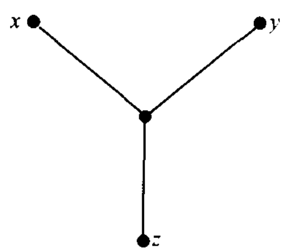


图 6.3　3 个对象的惟一一棵树

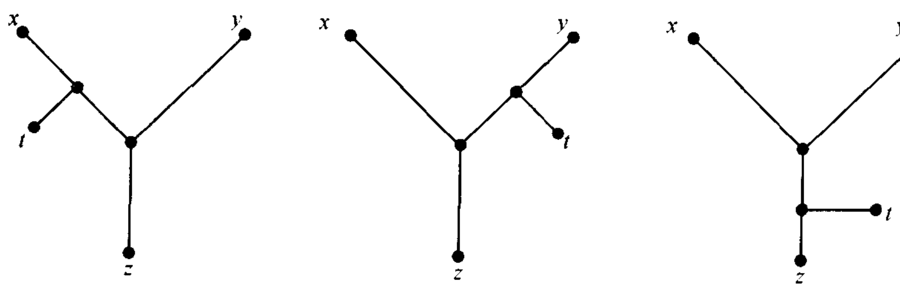


图 6.4　4 个对象有 3 棵树

现已知道，一个完全种系树问题是否是一个计算难题取决于性状的性质。如果性状是无序的，则计算问题是 NP-完全的；但如果性状是有序的，则它可以被有效地解决。以这两个根据作背景，我们进一步研究某些特殊实例的更多细节，这些实例代表了性状状态

182 矩阵算法结果的一个范例。

6.2 二值性状状态

我们首先研究完全种系树这个特殊例子，其中的性状是二值的，如表6.1和表6.2所示。注意，在这种情况下，有序和无序性状的区别取决于对性状是否有向的了解，这意味着如果我们知道状态1是祖先状态和状态0是衍生状态(或反之亦然)，那么性状就是有序的，反之则是无序的。正如我们将看到的，对二值性状(有序的或无序的)，完全种系树问题可被有效地解决。我们在本节展示一个运行时间为$O(nm)$的算法。

将介绍的算法分两阶段工作(实际上这两阶段可结合执行，但分开解释更为清楚)。在第一阶段，算法要判定输入矩阵是否能容纳一个完全种系树；如果能，算法的第二阶段将构造一个可能的种系树。

在将要介绍的算法中，假定状态0是祖先状态，状态1是衍生状态。随后将看到如何放宽这一限制。注意，由于处理的是有向二值性状，一个有根树T(它是所输入矩阵M的完全种系树)将有下面的前已述及的性质：输入矩阵M的每一个性状对应于T中的一条边，这条边标记了这个性状从状态0到状态1的转变。这样的边由其对应的性状标记。根总是具有性状状态向量$(0,0,\cdots,0)$，这意味着当我们遍历一条从叶对象i到树根的路径时，对于沿途边的标记所对应的性状，对象i的状态均为1(图6.2)。

在这个特殊例子里，存在一个很简单的充分必要条件，它能使我们知道对一个给定矩阵M是否存在一个完全种系树。为了陈述这个条件，先需要几个定义。

M的每一列j是一个性状，所以我们将对术语列和性状不加区别。M的每一行i是一个对象，所以使用行和对象也不加区别。

定义6.1 对M的每一列j，令O_j是j的状态为1的对象的集合，$\overline{O}_j$是j的状态为0的对象的集合。

该充分必要条件如下：

引理6.1 一个二值矩阵M有一个完全种系树，当且仅当对每一对性状i和j集合O_i和O_j不相交，或其中一个包含另一个。

证明：假定M有一个完全种系树。因为M是二值的，对每一个性状i，我们能够惟一地将它与树中的一个边(u,v)相关联。而且，以v为根的子树(假定它是两个节点中较深
183 的一个)包含所有性状i的状态是1的节点，任何性状i的状态是0的节点均不属于该子树。现在假定有三个对象A、B和C，$A,B\in O_i$，$C\notin O_i$，$B,C\in O_j$，$A\notin O_j$。这意味着，根据性状i，A和B属于同一子树，但C不属于；而根据性状j，B和C又属于同一子树，这相互矛盾。

现假定M的所有列的对都满足引理的条件，下面将归纳性地展示如何构造有根完全种系树。假设只有一个性状，比如说1，这个性状将对象分为两个集合，$A=O_1$，$B=\overline{O}_1$。创建一个根，再创建集合A的一个节点a以及集合B的一个节点b。用标记为1的边将节点a连接到根，用没有标记的边将节点b连接到根。最后一步是，根据集合中的对象将根的每个孩子分裂成尽可能多的叶，这样创建的树显然是完全种系树。这是基本

的情况。然后假定对 k 个性状创建树，但并不执行最后一步(也就是说没有叶子，节点仍含有对象的集合)。现在，我们处理第 $k+1$ 个性状，这个性状同样能分割对象集合，通过它可从树 T 中得到树 T'。只要由性状 $k+1$ 导出的分划能分割属于同一节点的对象，我们就能做到这一点。如果情况不是这样，则我们不得不用性状 $k+1$ 标记两条边，其结果是种系树不是完全的。但这种情况不会发生。假定它发生了，即假定性状 $k+1$ 分割了分别属于节点 a 和 b 的对象，因为这些是不同的节点，必然存在一个性状 i 在开始时将它们归入不同的节点，这就是 $O_i \cap O_{k+1} \neq \phi$ 的情况，因为要么 a 中的要么 b 中的对象属于 O_i。但是也有这种情况，即 O_i 既不包含 O_{k+1} 也不被包含在 O_{k+1} 中，因为不管 a 中的对象还是 b 中的对象都不在 O_i 中，这与我们的假设矛盾，因此引理成立。 ■

从这个引理容易看出为什么表 6.1 中的性状状态矩阵不包含一个完全种系树：c_1 列与 c_5 列并不满足引理的条件，我们称这些性状不**相容**(compatible)。性状之间相容性的概念在此可有更精确的意义，它也是引理 6.1 的另一种陈述，即一组二值有向性状是相容的，当且仅当它们是成对相容的(pairwise compatible)，而成对相容性由引理所陈述的集合关系确定。

我们提到二值情况的算法是两阶段的，第一阶段是确定矩阵是否能容纳一个完全种系树，第二阶段构造一个种系树。引理 6.1 已给出第一阶段的算法，即依次考虑每一列，检查其是否与所有其他列相容。每一次检查需耗时 $O(n)$，共须进行 $O(m^2)$次检查，因此总的运行时间为 $O(nm^2)$。根据这个引理的证明，有可能发展出一个构造树的算法，其时间复杂性相同。但我们的目标是给出一个运行时间为 $O(nm)$的算法。

如何达到这个目标？在判定阶段，每当算法检查一个新列时，亦扫描其余所有的列。为达到所需要的效率，我们仅能扫描所有的列一次(也许对列的安排施加某些次序)。这是一种什么次序呢？一个想法是首先处理被最多的对象拥有状态 1 的性状。直觉上，一旦我们处理完这个性状，就可以“忘记”它，因为所有其他性状要么是其子集要么不与其相交，而我们正是这么做的。图 6.5 给出这个算法。

现在证明这个算法是正确的。首先注意到，若 $O_j \subseteq O_k$，则 M 中 k 在 j 的左侧，即 $k <$ 184
j。假定 M 有一个完全种系树但算法告诉我们没有，即 M 有列 j 使得 $L_{ij} = k$ 和 $L_{lj} = k' < k$，这意味着(查看列 k)元素 $M_{lk} = 0$ 而 $M_{ik} = 1$，因此 $O_k \cap O_j \neq \phi$。另一方面，没有集合是另一个集合的子集：O_j 不是 O_k 的子集，因为 M_{lk}是 0 而 M_{lj}是 1；而 O_k 也不是 O_j 的子集，因为 k 在 j 的左侧。所以，k 和 j 不满足引理 6.1，而这是矛盾的。

我们现在断言，如果算法告诉我们存在一个完全种系树，则这个种系树即真正的种系树。下面将显示，对 M 中的每一对列引理 6.1 都是满足的。考虑任意列 j 和在 j 左侧的列 k，$O_j \subseteq O_k$，且 k 是最大值(即 k 是 L 的列 j 的公共非零值)。根据这个定义，k 和 j 之间的任何列 p 都具有性质 $O_p \cap O_j = \phi$。如果 $k > 0$，我们可应用一样的论断表明，O_j 与所有位于 k 与 k'之间的其他列 q 的交集是空的(这里 k'是 L 中列 k 的公共非零值)。我们因而说明，对任意列 j，对应的 O_j 被包含于其左边的某些集合中，且与其他集合的交集为空，这正是要证明的内容。

算法的运行时间为 $O(nm)$，因为 M 的基数排序耗时 $O(nm)$(每一列耗时 $O(n)$)，其他步骤可轻易地在同样的时间内完成。 185

```
Algorithm Perfect Binary Phylogeny Decision
  input: binary matrix M
  output: TRUE if M admits a perfect phylogeny, FALSE otherwise
  // Assumes all columns are distinct
  Sort columns based on the number of 1s in each column using radix-sort
     (columns with more 1s come first)
  // Initialize auxiliary matrix L
  for each L_ij do
     L_ij ← 0
  // Compute L
  for i ← 1 to n do
     k ← -1
     for j ← 1 to m do
        if M_ij = 1 then
           L_ij ← k
           // k is the rightmost column to the left of j such that
           // M_ik = 1. If no such column exists, k = -1
           k ← j
  // check columns of L
  for each column j of L do
     if L_ij ≠ L_lj for some i, l and both L_ij and L_lj are nonzero then
        return FALSE
  return TRUE
```

图 6.5 判定一个有向二值性状状态集合是否包含一个完全种系树的算法

==========构 造 阶 段==========

构造树的思想是依次处理对象,在处理中构造树。每一棵树均从树根开始构造。对每一个对象,查看状态是 1 的性状。每当发现没有边被这样的性状标记,就创立一个新节点,把它与当前节点用边相连,用这个性状标记边,新节点变为当前节点。如果已有一条边标记有此性状,则继续下一个性状,让当前节点成为这个边的末端。该算法的形式描述见图 6.6。注意所获得的树是有根的,但通过移去根和将根的孩子节点相连即可获得无根树。下文将说明该算法构造的树是一个 M 的完全种系树,运行时间为 $O(nm)$。

首先回想一下,M 中的每个性状必须对应且仅对应树的一条边,这正是算法所做的。另外,在一个从根到叶(对象)i 的通路中我们仅遍历某些性状所对应的边,对这些性状 i 的状态是 1,而这也是算法所做的。如果对 M 中的每一个元素仅查看一次,且每一个元素仅花费常数时间,则运行时间为 $O(nm)$。对表 6.2 的矩阵应用该算法,可获得图 6.2 所示的种系树。

我们曾提及,当性状是无序的时仍可有效地解决问题。事实上,如果施行一个简单的
186 转换,同样的算法即可奏效。当性状是无序的时,考虑字母 a 和 b 表示的状态。转换过程如下:对每一个性状,确定发生较多的状态,把它变为 0,把其他状态变为 1。如果两种状态发生的频率均等,则可任选一个为 0,另一个为 1,然后即可应用上述算法。这样做可行的原因来自于下面的事实:

事实:对于一组具有两个二值无序性状 i 与 j 的对象,存在一个完全种系树,当且仅当

```
Algorithm Perfect Binary Phylogeny Construction
    input: binary matrix M with columns sorted in nonincreasing order by number of 1s
    output: perfect phylogeny for M
    Create root
    for each object i do
        curNode ← root
        for j ← 1 to m do
            if M_ij = 1 then
                if there already exists edge (curNode, u) labeled j then
                    curNode ← u
                else
                    Create node u
                    Create edge (curNode, u) labeled j
                    curNode ← u
        Place i in curNode
    for each node u except root do
        Create as many leaves linked to u as there are objects in u
```

图 6.6　由有向二值性状构造完全种系树的算法

O_i，O_j 和 $\overline{O}_i$，$\overline{O}_j$ 均满足引理 6.1 中的条件。

先前的转换保留了上述事实所要求的无序性状之间的相容性，我们把这个留作习题。

现在以两个事实结束本节。第一个是，不存在比上述算法更快的算法。很清楚，任何求解该问题的算法都必须至少查看一次矩阵中的每一个元素，这得出时间下限 $\Omega(nm)$。这意味着上面给出的算法是优化的。但是，如果用每一个对象的性状表作输入，则有几个算法可解决该问题，其运行时间与性状表的长度成比例，即 $O(n+m)$。第二个事实是，对于无序性状，当状态的数量是固定的时，已知存在多项式时间的算法。对于这两个事实的细节，请参见本章末的文献提要。

6.3　两个性状

这里我们研究完全种系树问题的另一个特例，允许更一般化的条件，即性状可以是无序的且可以有任意数量的状态，但要限定性状的最大数量。作为特例，我们研究只有两个性状的情况。

如同二值状态的例子一样，显而易见仅有两个性状将使计算很容易。下面介绍该问题的一个简单算法，此算法应用了图论中的非平凡结果。现以几个定义开始。

定义 6.2　三角剖分图(triangulated graph)是无向图，图中任何有 4 个或更多个顶点的环有一条弦，即连接环中两个不连续顶点的边。

回忆先前的结论，即一个子树是一个树的连通子图，以及对某一给定性状具有相同状态的树节点集合构成完全种系树中的一个子树。三角剖分图与完全种系树之间的联系由下面的定理给出。

定理 6.1　对树 T 的每个子树集合 $\{T_1, T_2, \dots, T_l\}$，均存在对应的三角剖分图，反之亦然。 187

我们将不介绍此定理的证明，但将描述对应于其子树集合的图，这个图是一个方便的

工具，帮助我们确定一组性状是否相容（即对于对应的对象是否存在一个完全种系树）。下面将更清楚地说明，当且仅当根据上述定理构造的图可以被三角剖分时，一组性状是相容的。为理解该图我们需要更多的定义。

定义 6.3　图 G 是一组集合 C 的一个交集图（intersection graph），在该图中，C 的每一个集合对应于 G 的一个顶点；且用一条边连接 G 中的两个顶点，如果顶点对应的集合有一个非空的交集。

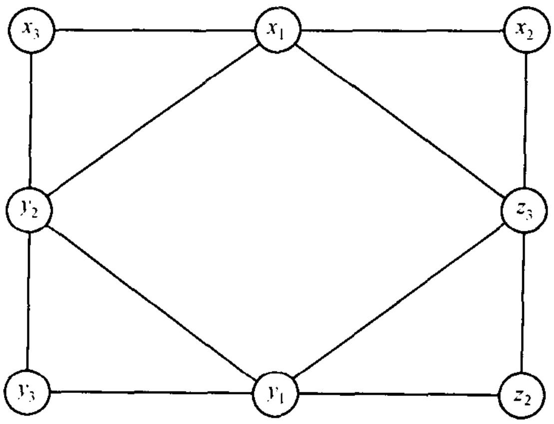


图 6.7　对应于表 6.3 中矩阵的 SIG，每一个顶点标记的下标代表顶点的颜色

定理 6.1 的证明中的图是一个交集图。为描述交集图，我们首先介绍另一种图，状态交集图（state intersection graph，SIG），它是一个中间步骤。已知一个性状状态矩阵，其 SIG 可通过下面的方式得到。对每个性状的每个状态，在 SIG 中创立一个顶点。注意每一个状态（即每一个顶点）对应一个集合，即对该性状具有该状态的对象的集合。然后，在 SIG 中的顶点 u 和 v 之间创建边，当且仅当这些顶点对应的集合有一个非空交集。这就是为什么这种图叫做状态交集图的原因。图 6.7 是从表 6.3 矩阵获得的状态交集图的例子。

表 6.3　一个对象×性状矩阵

	性　状		
对　象	c_1	c_2	c_3
A	x_1	x_2	z_3
B	x_1	y_2	x_3
C	y_1	y_2	y_3
D	y_1	z_2	z_3

注意，定理 6.1 论及的图是根据一个子树的集合构造的，前一部分描述的图则是根据一个对象的集合构造的，对象是种系树中的叶。为获得定理中所述的图，我们必须通过增加新的边“放大”SIG 以解释内部节点。我们将在这个放大的图中检查三角剖分性
188 质。实际上，我们试图为 SIG 增加边来令其三角剖分化。不过，边不能随意地增加，而必须根据每一个性状来考虑对象之间的关系。为进一步解释这点我们还需要一个定义。

定义 6.4　给定一个在 V 上着色 c 的图 $G=(V,E)$，如果存在一个三角剖分图 $H=(V,E')$，使得 $E'\subseteq E$，c 是 H 的一个有效着色，则说 G 可被 c-三角剖分（c-triangulated）。换言之，任何出现在 E' 中但不出现在 E 中的边用不同的染色连接两个顶点。

现可观察到，由于一个对象不可能有同一性状的两种不同状态，性状状态矩阵的性状集定义了 SIG 的一个着色，我们因而获得了定理 6.1 的一个新公式，它可澄清 SIG 和种系树之间的连接。

定理 6.2　带一个定义着色 c 的性状集的性状状态矩阵 M 包含一个完全种系树，当且仅当其对应的 SIG 可被 c-三角剖分。

在图 6.7 的例子中，每一个顶点的着色由性状给出，该性状是顶点标记的一部分。为

了三角剖分该图,我们只需创建顶点 y_2 和 z_3 之间的一条边。这是可能的,因为这两个顶点有不同的颜色[但我们不能创建边(x_1, y_1),因为二者具有同样的颜色]。这意味着表 6.3 中的矩阵包含一个完全种系树。

注意定理 6.2 的普适性:它完全未提及性状的数量。回想一下,无序性状的完全种系树问题一般是 NP-完全的,所以不应奇怪图的 c-三角剖分同样是一个 NP-完全问题。但若 M 仅有两个性状,则 SIG 仅有两种颜色,由此产生下面的结果。

定理 6.3　一个仅有两个性状的性状状态矩阵 M 包含一个完全种系树,当且仅当其对应的 SIG 是无环的。

证明:首先注意到,如果 SIG 是无环的,它就已是 c-三角剖分的,因此,根据定理 6.2,M 包含一个种系树。为了证明"仅当"部分,使用图论的一个经典结果就足够了,即当且仅当图的所有环的长均为偶数时,该图可用两种颜色着色。从定理 6.2 可以知道 SIG 可被 2-三角剖分,这意味着可用两种颜色着色,所以 SIG 不能有奇数长度的环。三角形是一个奇数长度的环,因此,SIG 能满足 2-三角剖分性质的惟一途径是它是无环的。 ■ 189

根据上面的定理,我们现在有一个简单有效的算法来检查一个两性状状态矩阵是否包含一个种系树,因为对一个有 $|V|$ 个顶点和 $|E|$ 条边的图,无环性测试仅需耗时 $O(|V|+|E|)$。在本例中,顶点的数量是 $O(n)$(因为只有两个性状,所以最多只有 $2n$ 个状态),边的数量也是 $O(n)$,因此算法有线性的运行时间。重构完全种系树 T(如果它存在的话)的过程如下:创建辅助图 $G=(V,E)$,使得对 SIG 的每一条边 V 中都存在一个顶点。这样,对于顶点 $v\in V$[它对应于 SIG 中的边(x,y)],存在一个对象的集合 L_v,它是 x 集合和 y 集合的交集(回想一下,SIG 中的一个顶点对应于一个对象集合)。如果 $L_u\cap L_v\neq\phi$,则在两个顶点 $u,v\in V$ 之间创建一条边。这棵期望的树是 G 的任意生成树(这意味着这种树可以不是惟一的),L_v 中的对象作为叶子加到每个节点 v 上。所产生的树是无根的,并可在 $O(n)$时间内得到。这种构造仅适用于两个性状的例子。如果存在多个性状,树中的节点将对应于三角剖分 SIG 中的最大完全子图(亦即最大圈)。这个事实来自定理 6.1 的证明,这里从略,它也解释了如何连通所获得的节点以构造树。

一个构造二性状树的例子见图 6.8,该树是由表 6.4 中的矩阵获得的。通过检查对给定性状具有相同状态的所有对象是否构成一棵子树,可核实该树是否为一个完全种系

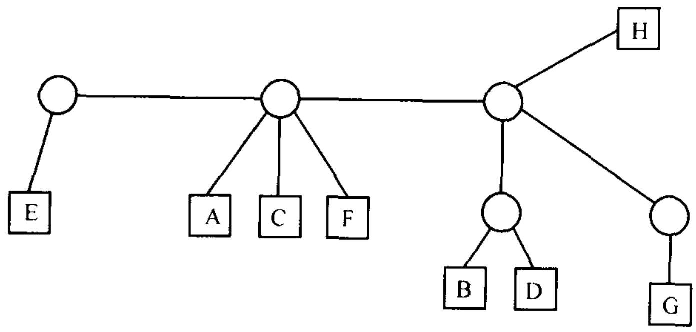


图 6.8　由表 6.4 的数据重构的树

表 6.4　双性状状态矩阵的一个例子

	性　状	
对　象	c_1	c_2
A	x_1	x_2
B	y_1	y_2
C	x_1	x_2
D	y_1	y_2
E	w_1	x_2
F	x_1	x_2
G	z_1	y_2
H	x_1	y_2

树。例如，对性状 c_1，对象 A、C、F 和 H 都有状态 x_1，并确实形成子树；相反，E 和 G 不形成子树，见习题 9。

所得到的结果为下面更一般的结果提供了基础，即一个对 m 的任意固定值有多项式运行时间的算法（换言之，m 是运行时间函数中的一个阶）。对这个结果的描述超出了本书的范围，但在文献提要中给出了
190 参考。

6.4　种系树的简约性和相容性

从前几节可以看到，虽然对无序的性状完全种系树问题是 NP-完全的，但一些参数固定的特殊例子是多项式时间可求解的。参数固定的算法在实践中看起来相当好，但问题是真正的性状状态矩阵可能并不包含完全种系树，原因是两方面的，其一，生物学实验数据总存在错误；其二，算法所依据的假设（无反转和无趋同）有时不成立。基于这些原因，研究其他重构种系树的方法是必要的。

如果忽略错误但仍遵循假设，则存在两类方法。第一是允许反转和趋同，但尽量减小它们的发生率，这称为*简约性标准*（parsimony criterion）；第二是坚持避免反转和趋同，排除导致这些"问题"的性状。当然，排除的性状越少越好。这意味着在该算法中我们要找出性状的一个最大集合，从中可构造完全种系树（或者说，相容性状的最大集合）。这称为*相容性标准*（compatibility criterion）。

这两种标准都导致优化问题，这类问题通常比判定问题更难，如在 6.1 节所考虑的完全种系树问题。因为我们已知，对无序性状判定完全种系树是一个 NP-完全问题，刚才讲的这些问题至少同样难，因此它们也是 NP-完全问题。但也许我们会幸运地遇上有序性状的例子，而如 6.1 节所示，这种情况的完全种系树问题是多项式时间可解的。不幸的是，没有这样的好运气，上文所述的两个标准对于有序性状也导致 NP-难题。下面将详细讨论这些问题。

当应用简约性标准时，我们尽量减少反转和（/或）趋同的事件。通常，这意味着在重
191 构的种系树中要减少*状态转换*（state transition，即边）的数量。如果我们允许反转但禁止趋同，则得到所谓的 Dollo 简约性标准；另一方面，如果禁止反转但将趋同事件最小化，则得到 Camin-Sokal 简约性标准。这两种标准都导致 NP-难题，其证明基于 Steiner 树问题，这是图论中的一个经典的优化问题。

简约性的概念也出现在其他问题的文献中。例如，已知一个对象集合以及将这些对象相关联的树，将状态赋予内部节点使得节点之间具有最少数量的状态转换。该问题的有效算法是否存在依赖于问题的确切特征，我们不讨论这样的算法，而仅在本章末给出参考文献。

现将注意力转向相容性标准，并给出有关的判定问题是 NP-完全问题的证明。该证

明用到了圈问题(clique problem),它是众所周知的 NP-完全问题。事实上,我们表明圈和相容性是等价的,这意味着用于圈问题的枚举算法可用于相容性问题,且存在着利用这一点的种系树软件包。毫无疑问,能够应用这种途径求解的相容性问题的规模是有限的,不过,该方法看起来仍然有用。

相容性问题是 NP-完全问题的一种证明如下:

论题:相容性

实例:有 n 个对象和 m 个有向二值性状的性状状态矩阵 M,以及一个正整数 $B \leqslant m$。

问题:是否存在一个性状的子集 L,它满足引理 6.1 且 $|L| \geqslant B$?

注意,我们正在处理该问题的一个非常受限的形式,因为性状是二值的和有向的。由于这个限制,相应的相容性概念正是在引理 6.1 中描述的。如前所述,NP-完全性的证明用到圈问题,它可形式地描述如下:

论题:圈

实例:图 $G=(V,E)$ 和正整数 $K \leqslant |V|$。

问题:G 是否含有子集 $V' \subseteq V$, $|V'| \geqslant K$,使得 V' 中的每一对顶点都由 E 中的一条边相连?

我们要表明相容性问题是 NP-完全的且等价于圈。如 2.3 节所描述的,这要求我们证明两个问题属 NP 类,并可在多项式时间内相互转换。由于已假定圈问题是 NP-完全问题,所以立即得到它属于 NP 类,其余部分由以下定理给出。

定理 6.4　相容性是 NP-完全的。

证明:给定相容性问题的一个解,可容易地检查实际上它是否用 6.2 节描述的多项式时间算法求解了该问题。因此我们可看到相容性是属于 NP 的。我们现在展示如何在多项式时间内将圈问题转换成相容性问题。

已知一个圈 $G=(V,E)$,我们按下面的方法构造相容性的一个例子 M。令 $m=|V|$,即对 V 中的每一个顶点 v_i,我们在 M 中创建一个性状 i。M 中对象数量(行 192
数)为 $n=3m(m-1)/2$,也就是说对每一对可能的性状我们创建 3 个对象。现在必须填充 M。对每一对不是 E 中边的(v_i, v_j),我们创建 M 中的 3 个对象 r、s 和 t,对 M 中的相应元素取下列值:$M_{ri}=0, M_{si}=1, M_{ti}=1, M_{rj}=1, M_{sj}=1, M_{tj}=0$。容易看出按这些值性状 i 和 j 并不兼容(见引理 6.1)。M 中剩下的元素应该是 0。上述的转换显然可在多项式时间内完成。

现在必须证明 G 含有圈 V',且 $|V'| \geqslant K$ 当且仅当 M 包含一个相容的性状子集 L, $|L| \geqslant K$。首先假定这样的一个圈是存在的,然后这个圈中的每一条边对应有 M 中的一对性状,使得每当性状之一在一个对象的状态是 1 时,在其他对象的状态便是 0(或二者均为 0)。所有这些性状对因而满足引理 6.1,它们是相容的。现假定 L 存在, $|L| \geqslant K$,则对于 L 中的每一对性状 V 中有一对由边相连的顶点。所有这些顶点对和起来形成一个大于或等于 K 的圈。 ■

定理 6.5　相容性问题可在多项式时间内转换为圈问题。

证明:给定一个相容性问题的实例 M,按下面的方法获得一个圈的实例 $G=(V,E)$:对 M 的每一个性状创建 V 中的一个顶点。当且仅当性状 i 和 j 相容时,联合两个顶点 v_j

和 v_i。性状对之间相容性的检测可容易地在多项式时间内完成。引理 6.1 确保了当且仅当 M 含有相容性状的一个规模等于或大于 B 的子集时，G 含有一个规模等于或大于 B 的圈。■

6.5 距离矩阵算法

本节我们考虑基于 n 个对象之间数值数据比较的树重构问题。基本输入是一个 $n\times n$ 矩阵 M，其中元素 M_{ij} 是非负实数，叫做对象 i 和 j 之间的距离，这种距离来自于对象之间的比较。例如，如果对象是 DNA 序列，则可使用 3.6.1 所描述的序列间距离的概念。

第一个要考虑的问题可非形式地陈述如下：给定距离矩阵 M，要构造一棵边带权的树，其中每一片树叶对应于 M 中的一个对象，使得测得的叶 i 和 j 之间的距离恰对应于 M_{ij} 的值。当这样的树能被构造时，我们说 M 中的距离是*可加的*(additive)。

我们将会看到，实际中距离矩阵很少是可加的，而且将缘于可加性的偏差最小化的努
193 力将导致 NP-难题。这引发了我们要考虑的第二个问题，即用两个矩阵而不是一个矩阵作为构造树的输入，这些矩阵给出了对象对之间距离的上限和下限，而问题就是要找出一个在上、下限之间的树。

6.5.1 重构可加树

当讨论距离时，我们需要*度量空间*(metric space)的概念。早在 3.6.1 中我们就已经见过这个概念，当时叫做距离。为方便起见这里重复这个定义。度量空间是对象的集合 O，对每一对 $i,j\in O$ 有一个非负实数 d_{ij} 与下列性质相关联：

$$d_{ij}>0 \text{ 对于 } i\neq j \tag{6.1}$$

$$d_{ij}=0 \text{ 对于 } i=j \tag{6.2}$$

$$d_{ij}=d_{ji} \text{ 对所有的 } i \text{ 和 } j \tag{6.3}$$

$$d_{ij}\leqslant d_{ik}+d_{kj} \text{ 对所有的 } i、j、k \tag{6.4}$$

我们要求输入矩阵 M 是一个度量空间，这意味着它是个对称矩阵(根据性质 6.3)。现在我们必须确切定义究竟什么是基于 M 构造一棵树。

因为 M 是 $n\times n$ 矩阵，故树必须有 n 个叶。叶是 1 度的节点，所有其他节点的度必须是 3，这样的树因而是无根树。树中所有边均有非负权值(这个要求以及对度数的要求意味着，与其有一个度数为 4 的内节点，不如把它分为两个度数为 3 的节点，每个节点有一个长度为 0 的边)。树上节点 u 和 v 之间通路的权等于构成此通路的各边权之和。最后的要求是，任何两个叶子 i 和 j 之间通路的权必须等于 M_{ij}。如果能找到这样的树 T，则称 M 和 T 是可加的。

基于这个框架，我们的基本问题是：给定矩阵 M，它是一个度量空间，该矩阵是否可加？如果是可加的，如何构造对应的可加树？第一个问题的答案由下面的引理给出。

引理 6.2 度量空间 O 是可加的，当且仅当任给 O 的 4 个对象，我们用 i、j、k 和 l 标记它们时能满足

$$d_{ij} + d_{kl} = d_{ik} + d_{jl} \geqslant d_{il} + d_{jk}$$

我们将不介绍此漂亮结果的证明(这称为著名的 4 点条件),读者可参考文献提要。我们将专注于回答第二个问题,即构造可加树的算法。这是一个多项式时间的算法,理解它(同时也是设计它)的最好方式是归纳。

现假定矩阵 M 已满足引理 6.2 的条件,且假定对象的数量 n 是 2(x 和 y),那么容易构造一个树:仅仅用一条带权边 M_{xy} 将节点 x 和 y 连接起来。假定加入第 3 个对象 z,如何在树中将 z 包括进去?很清楚我们应该获得看起来像图 6.9 中的树。问题是节点 x 和 y 之间的中心节点 c 的确切位置在何处,或者说,边(z,c)、(x,c)和(y,c)的长度应是 194
多少?以 d_{ij} 标记边(i,j)的长度,可获得下列等式:

$$M_{xz} = d_{xc} + d_{zc} \tag{6.5}$$

$$M_{yz} = d_{yc} + d_{zc} \tag{6.6}$$

另一方面,我们知道,

$$d_{yc} = M_{xy} - d_{xc} \tag{6.7}$$

如果从等式(6.5)减去等式(6.6),并用等式(6.7)代替 d_{yc},可得到:

$$d_{xc} = \frac{M_{xy} + M_{xz} - M_{yz}}{2} \tag{6.8}$$

注意,由于三角不等式(式 6.4),$d_{xc} \geqslant 0$,同样地可处理其他两个未知量,得到:

$$d_{yc} = \frac{M_{xy} + M_{yz} - M_{xz}}{2} \tag{6.9}$$

$$d_{zc} = \frac{M_{xz} + M_{yz} - M_{xy}}{2} \tag{6.10}$$

我们现在得到了对应于 3 个对象的可加树。注意该树是惟一的,正如两个对象的树一样。现在让我们加上第 4 个对象 w。(我们查看归纳的这么多基例的原因是,我们将在后面使用它们,以说明对于任意数量的对象可加树都是惟一的)。对象 w 将创立另一个内节点 c_2,现在的问题是判定 c_2 是否应分裂边(x,c)、(y,c)或(z,c)。我们选择已存在于该树中的任何两个节点,如 x 和 y,并应用等式(6.8)、(6.9)和(6.10),以 w 替换 z。如果我们发现节点 c_2 应该分裂(x,c)或(y,c),则完成之;但如果发现 c_2 恰巧与节点 c 重合,这意味着边(z,c)应是被分裂的边,因此不得不再一次应用同样的等式。这次使用 z 并以 w 替换 x(或 y)。如果 4 个对象有两个可加树会怎样?在这种情况下,可把 c_2 放在两个不同的边上,例如(x,c)和(z,c),但这又意味着对 3 个对象 x、z 和 w 有两个不同的树,这与我们早先的结果相矛盾,即任何 3 个对象只能有一个可加树。因此,我们推断 4 个对象同样只能有一个树。

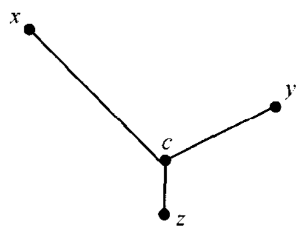


图 6.9　3 个对象的可加树

现在我们假设已构造了一个 k 个对象的树,在树上增加对象 $k+1$ 的过程类似于上面增加第 4 个对象的情况。我们选择已在树上的任意两个对象 x 和 y,并计算新的内节 195
点的位置。如果这个新节点跟任何现存的节点不重合,则构造完成,亦即我们确切地知道了需被分裂的边以将对象 $k+1$ 连接到树上;否则,分裂位置将落在一个现存的节点上,

比如 u。因为 u 是一个内节点,我们知道存在一个子树(可以是仅有一个叶子的子树)挂在这个节点上。现在选择属于这个子树的任意对象,比如 r,并对对象 x(或 y)、r 和对象 $k+1$ 应用等式(6.8)、(6.9)和(6.10)。我们重复该过程直到找到正确的分裂位置。

如果该树存在,则这个算法清楚地构造了该可加树。对加入到树中的每一个对象,或许需检查所有已存在的对象,每次检查花费常数时间。这意味着在最坏的情况下算法的运行时间为 $O(n^2)$。

例 6.1　为表6.5中的对象构造可加树。我们由创建边(A,B)开始,其长度为63。再加入对象 C。应用公式(6.8)、(6.9)和(6.10)可发现,在距离 A 为 39 距离 B 为 24 的地方应创建一个新的节点 x_1,然后增加边(x_1,C),其长为 55。现在考虑对象 E。如对已存在于树上的两个对象 A、C 应用这些公式,我们发现 E 应与 B 属于同一子树,因此必须对 E、B 和 A 应用公式。由此我们发现新节点 x_2 的位置,距离 x_1 是 18,相距 B 是 6。以此方式做下去,则可最后得到图 6.10 所示的树。

表 6.5　例 6.1 的输入矩阵 M

	B	C	D	E	F	G
A	63	94	111	67	23	107
B		79	96	16	58	92
C			47	83	89	43
D				100	106	20
E					62	96
F						102

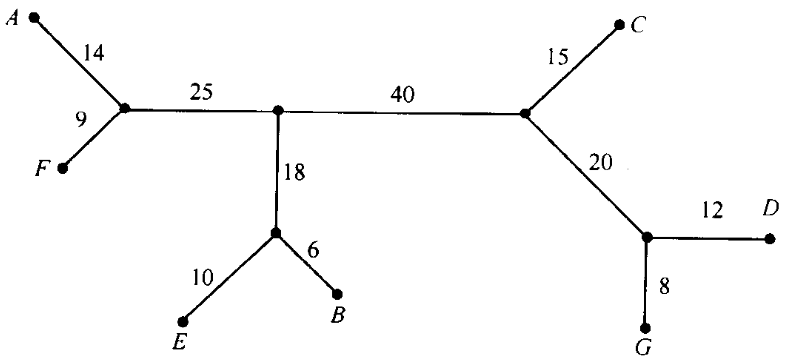


图 6.10　例 6.1 的可加树,边长已标明

现在我们结束对惟一性的讨论,结论是,当 $n>4$ 时,对于可加树,不论是其拓扑结构还是边的长度都是惟一的。首先看一看拓扑结构。任意三个树叶决定一个内节点,这个内节点反过来又决定了所有叶的一个惟一分划 $\{P_1, P_2, P_3\}$。如果存在两个不同的拓扑结构,则必然有一个三个树叶的集合,比如 x、y 和 z,决定了第一个树中的分划 $\{P_1, P_2, P_3\}$ 和第二个树中的分划 $\{Q_1, Q_2, Q_3\}$,使得这两个分划是不同的。那么就会有这样的情况,即会有一个树叶,比如 x,既属于 P_1 又属于 Q_2,而 $P_1 \neq Q_2$。这接下来意味着必然
196 有某个对象,比如 w,属于 P_1 但不属于 Q_2。这等于说对四个对象 x、y、z 和 w 有两个不同的树,这显然不能成立。

现在我们要证明边的长度是同样的情况。任何连到叶 x 的边必须有惟一的长度,因为由 x 和树中另外两个其他对象所确定的树是惟一的。内部的边同样是惟一的,因为它

们把树叶划分成集合 P_1、P_2、P_3 和 P_4。从每一个集合中取出一个叶，则得到 4 个对象，而我们知道用这 4 个对象只能构造一个树，由此内部边的边长也是惟一的。

我们已介绍了重构可加树的一个简单精致的算法，不过，这个结果在实践中用途有限，因为真正的距离矩阵很少是可加的。这种情况并不一定是由于距离度量的错误而导致的。有可能在要构造的树状的生物序列集中，祖先序列在同一位点发生多次改变(例如，DNA 中的某个碱基可从 A 变到 C 再变到 G)。换句话说，早期改变的记录被后来的改变破坏掉了。另外，趋同进化也可能发生。允许这些可能性将导致不可加的配对距离，这种情形尤其可能发生在蛋白质序列中。

这些事实使得我们对可加树问题归纳出一个结论。一种可能性是，获取一个与可加树尽可能接近的树。不幸的是，在“接近”的概念下，问题将变成 NP-难题。另一种处理该问题的方法富于技巧性，它正是下一部分的论题。

*6.5.2　重构超度量树

配对距离测量有时带来不确定性。假定这种不确定性可被定量，则这些测量值都可用区间来表示，区间确定了真实距离的下限和上限。根据这些数据，我们可以获得两个距 197
离矩阵，M^l 和 M^h，分别含有配对距离的下限和上限。现在重构种系树的问题是，构造一个进化树使得树上对象间的距离值要居于这两个输入矩阵“之间”。换言之，令 d_{ij} 为树上两个对象 i 和 j 之间的距离，则下面的不等式应成立：

$$M^l_{ij} \leqslant d_{ij} \leqslant M^h_{ij} \tag{6.11}$$

我们称式 6.11 为*夹层约束*(sandwich constraint)。本部分将展示，如果对期望的树施加一个附加的要求，树构造问题即可有效地解决。这个要求就是树必须是*超度量的*(ultrametric)。如果一个树是可加的，且具有一个根使得所有的叶到根路径的距离均相等，则此树为超度量树。这个附加要求的生物学意义是，被研究的对象以同等的速度从同一祖先进化而来。

因为超度量性质是可加性质的一个特例，真实的矩阵很少是超度量的。另一方面，仅仅简单地要求树“介于”矩阵 M^l 和 M^h 之间，问题就变得更加实际。下面我们将看到求解该问题的算法很有趣，它也是本章着笔最多的算法，因此算法的介绍将循下面的次序：首先给出一些定义和概念；然后在一个较高的层次介绍算法本身；接着是算法的细节，解释每一步是如何执行的以及给出运行时间分析；最后证明算法是可行的。

所需的第一个概念是把距离矩阵解释为无向有权边的图。已知 $n \times n$ 的距离矩阵 M，我们可把它解释为 n 个顶点的完全图，其中边 (i, j) 的权由 M_{ij} 给出。这个解释将在下面被广泛使用。我们假定每一对 i, j 都有一个 M_{ij}，因此对应的图总是连通的。对每一个 i 令 $M_{ii}=0$。基于上述解释我们讨论 G^l 和 G^h，它们分别是与上述 M^l、M^h 对应的图。边的权以函数 W 表示，如 $W(e)$ 或 $W(a, b)$，其中 e 和 (a, b) 是边。

我们所需的另一个概念是最小生成树 MST，这个概念已在 2.3 节中描述过。读者应该小心，不要把最小生成树与我们要建的树相混淆，后者是一个种系发生超度量树，前者仅是帮助我们解决问题的工具。我们以 U 表示超度量树，以 T 表示 MST。

MST T 定义在图 G^h 上。计算 T 的原因是它将指导树 U 的构造。特别是，对于 T

中的任意两个节点 a 和 b，从 a 到 b 的惟一通路中的最大权边被称作 a 和 b 的连接，用 $(a,b)_{\max}$ 表示。连接将用于定义下面关于 T 的边的关键性函数。

定义 6.5　G^h 的最小生成树的边 e 的削权(cut-weight)定义为：

$$CW(e) = \max\{M^l_{a,b} \mid e = (a,b)_{\max}\}$$

让我们理解这个定义的意义。给定矩阵 M^h，我们可以计算其对应图 G^h 的最小生成
198 树。每条边 $e=(a,b)\in T$ 是图 T 的至少一对顶点(a 和 b)的连接，它亦可是更多对顶点的连接。现取 e 是其连接的所有顶点对，在这些顶点对中，选择另一矩阵(即 M^l)的最大入口项。这就是 e 的削权。

关于削权作用的一个直觉认识有助于我们理解算法。考虑 U 中的叶节点 a 和 b，其间的距离 d_{ab} 是 U 中的最大叶间距离。很明显 d_{ab} 应大于 M^l 的最大入口项，此最大入口项记为 $\max^l$；但另一方面，d_{ab} 不需要比 $\max^l$ 大。因此我们必有 $d_{ab}=\max^l$，a 属于根的一个子树而 b 属于另一个子树。然后我们用 T 来确定两个子树之间剩余节点的一个划分。T 的性质和削权的定义保证了不同子树中的所有对象 x 和 y 有 $M^l_{xy}\leqslant d_{xy}\leqslant M^h_{xy}$，且所产生的树是超度量的。所有这些均在引理 6.4 和定理 6.6 的证明中给出。我们现在打算给出算法的初步草图。

1. 计算 G^h 的 MST T；
2. 对每一个边 $e\in T$，计算 $CW(e)$；
3. 建立超度量树 U。

第一步较简单，因为我们能应用已知的算法，花费 $O(n^2)$ 时间。其他两步更复杂，我们将详细讨论。

=========== 削 权 计 算 ===========

对每一对顶点 a 和 b，计算削权所需要的基本信息是 $(a,b)_{\max}$，我们通过构造另一个树(记做 R 树)来发现这些边。R 是一个有根的二叉树，对象均在叶子中。在 R 的每一个内节点中我们存记 T 的一个边。我们的目的是这样的：对每一对对象 a 和 b，作为 a 和 b 的**最低共同祖先**(lowest common ancestor)的 R 的一个内节点包含 $(a,b)_{\max}$。例如，在 R 的根中我们储存了 T 的最大权边，其理由是，对每一对对象 a 和 b，只要相对于最大权边 a 和 b 位于 T 中的对立侧，则该边就是 $(a,b)_{\max}$。我们可以利用这一事实设计一个算法以自顶到底的方式工作：首先创立 R 的树根，并用其储存最大权边 e；然后递归地执行同样的操作，先考虑 e 的一侧的节点(它们的根是总根的左侧孩子)；然后考虑 e 另一侧的节点(它们的根是总根的右侧孩子)。不过，自底向上构造 R 更有效，如图 6.11 所示的算法。这个算法使用了非相交集森林数据结构(见 2.3 节)，该数据结构用过程 *MakeSet*、*FindSet* 和 *Union* 表达。运行时间分析如下：排序边耗时 $O(n\log n)$；构造 R 耗时 $O[n\,\alpha(n,n)]$(因为使用非相交集森林数据结构)，所以总的运行时间为 $O(n\log n)$。

有了 R，计算削权就变得简单了。我们必须使用的基本操作是对任意两个树叶进行最低共同祖先操作。这是一个经典的计算机科学问题，存在着几个算法，可在 $O(n)$ 时间预处
199 理有 n 个叶的树，以便随后以常数时间进行最低共同祖先查询。算法的伪代码见图 6.12。计算削权所花费的总时间是 $O(n^2)$，因为我们要处理每一对对象，而每对的花费时间是常数。

```
Algorithm Construction of R
    input: minimum spanning tree T of G^h
    output: tree R
    for each object i do
        MakeSet(i)
        Create one-node tree for i
    Sort edges of T in nondecreasing order of weights
    for each edge e = (a, b) ∈ T in this order do
        A ← FindSet(a)
        B ← FindSet(b)
        if A ≠ B then
            // join trees
            r_a ← tree that contains a
            r_b ← tree that contains b
            create tree R
            R.edge ← e
            R.leftChild ← r_a
            R.rightChild ← r_b
            Union(A, B)
    return R
```

图 6.11　使用非相交集森林构造树 R

```
Algorithm Cut-weight Computation
    input: matrix M^l, trees T and R
    output: cut-weight array CW
    Pre-process R for lowest common ancestor (lca) queries
    for each edge e ∈ T do
        CW[e] ← 0
    for each pair of objects a and b do
        e ← lca(R, a, b)
        if M^l_ab > CW[e] then
            CW[e] ← M^l_ab
    return CW
```

图 6.12　削权的计算

==========构 造 U==========

我们现在介绍算法的最后部分，包括构造超度量树 U。我们将看到，这个构造过程本质上与构造 R 一样，只不过是用边的削权代替了边的权。我们用削权来排序 T 的边，然后，当根据更小的树构造 U 树时，我们用削权来确定 U 的边权。算法在图 6.13 中给出。 200

到此，读者可能会认为算法有些神秘。一会儿我们将提供算法可行的一个证明，这个证明包含了隐藏在算法后面的一些直觉。然而在证明之前我们先完成算法的运行时间分析。构造 U 花费的时间与构造 R 花费的时间相同，均为 $O(n\log n)$，因此算法总的时间是：对于 MST 计算为 $O(n^2)$，加上削权计算的 $O(n^2)$，再加上构造 U 的 $O(n\log n)$。$O(n^2)$决定了其他两项，因此它就是最终的运行时间复杂性。

```
Algorithm Construction of U
    input: trees R and T, array CW
    output: tree U
    for each object i do
        MakeSet(i)
        Create one-node tree for i
        height[i] ← 0   // auxiliary array
    Sort edges of T in nondecreasing order of cut-weights
    for each edge e = (a, b) ∈ T in this order do
        A ← FindSet(a)
        B ← FindSet(b)
        if A ≠ B then
            // join trees
            u_a ← tree that contains a
            u_b ← tree that contains b
            create tree U
            U.leftChild ← u_a
            U.rightChild ← u_b
            height[U] ← CW[e]/2
            W(u_a, U) ← height[U] − height[u_a]
            W(u_b, U) ← height[U] − height[u_b]
            Union(A, B)
    return U
```

图 6.13　构造超度量树 U

例 6.2　我们将根据表6.6的输入矩阵 M^l 和 M^h 计算一个实际例子。图 G^h 及其一个最
201 小生成树 T 在图 6.14 中;树 R 在图 6.15 中。从 R 中可引出的削权是:

$$CW[B,D]=1,$$
$$CW[A,C]=2,$$
$$CW[A,E]=3,$$
$$CW[A,D]=4.$$

所得的超度量树表示在图 6.16 中。

表 6.6　例 6.2 的输入矩阵 M^l 和 M^h

	M^l				M^h			
	B	C	D	E	B	C	D	E
A	3	2	4	3	7	3	6	5
B		4	1	1	10	4	8	
C			3	3		8	5	
D				1			7	

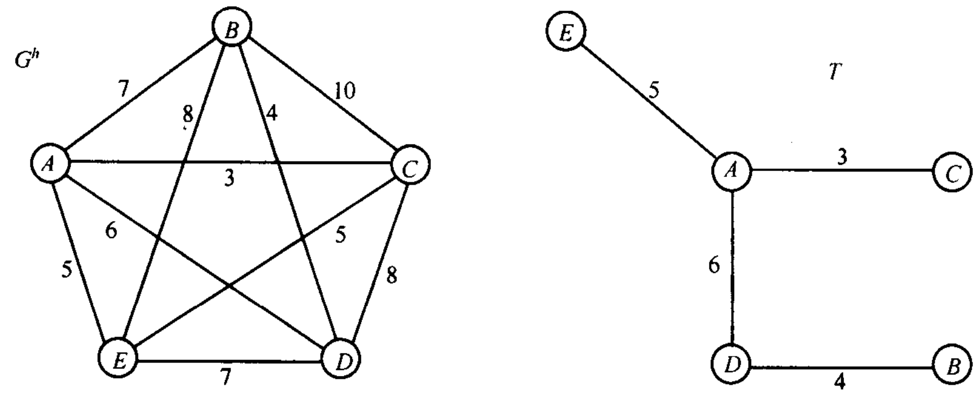


图 6.14　例 6.2 的图 G^h 和 MST T

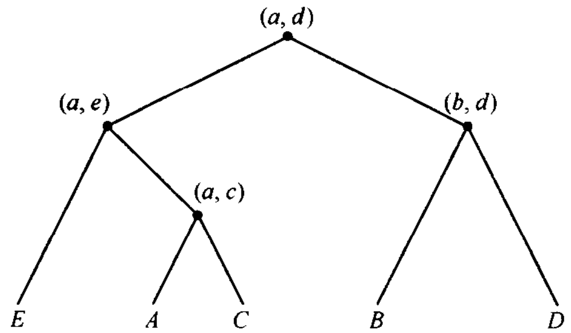


图 6.15　例 6.2 的树 R 202

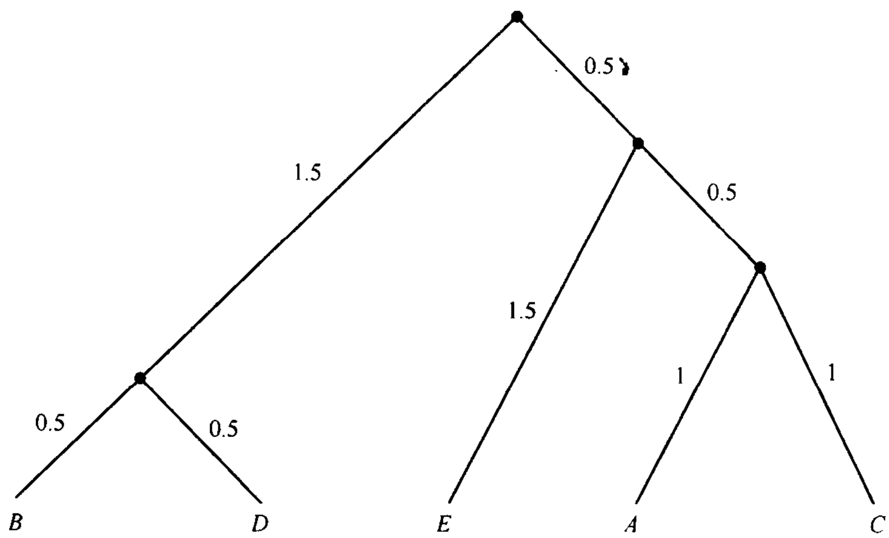


图 6.16　例 6.2 的超度量树 U,边的标记是节点间的距离

＝＝＝＝＝＝＝＝＝＝＝算法可行的证明＝＝＝＝＝＝＝＝＝＝＝

为了证明上文算法的正确性,我们需要更多地了解超度量距离。我们曾提及在超度量树中,根到叶的距离对所有的叶都相等。虽然这样的概念很容易掌握,但我们需要另外

的方法描述超度量距离，这由下面的引理给出。

引理 6.3　一个可加矩阵 M 是超度量的，当且仅当在对应的完全带权图 G 中，最大权边在任意环中都不是惟一的。

我们将该引理的证明留作习题 19，现在我们介绍算法可行的第一个证明。

引理 6.4　此算法构造的树 U 是超度量的。

证明：从 U 创建一个图 G，它只是个完全图，该图以 n 个对象作为顶点，以对象间的距离作为边权，对象间的距离由算法确定。现在考虑图中的一个环 C 以及环中的最大权边 (a,b)。这条边的权是某一边 $e\in T$ 的削权（这是算法工作的方式）。在这个算法中，当考虑 e 时会发生两种情况之一：要么 a 和 b 已属于同一子树（因此它们对应的子树已被汇合），要么它们属于不同的子树。对第一种情况，因为节点已属于同一子树，所以其在 U 中的距离已设定；因此情况必然是 C 中有另一条与 e 的权相等的边，它已在 e 之前被考
203 虑过，这满足引理 6.3。对第二种情况，对 e 的考虑导致 a 与 b 的首次汇合。因为 e 是 C 的最大权边，在检查了 e 之后，没有属于 C 的其他顶点对被汇合。但在这种情况下，C 中的另一个顶点对 x 和 y 也被首次汇合，否则，a 和 b 将是 U 的叶子且它们到 C 中其他顶点的距离仍将保持未定，而我们知道情况并非如此。因此，C 中 (x,y) 的权是 $CW(e)$，这也满足引理 6.3。

我们尚需证明距离满足夹层约束，下面将通过一个定理证明这一点，该定理事实上是本算法的核心。

定理 6.6　存在一个介于矩阵 M^l 和 M^h 之间的超度量树 U，当且仅当对每一对对象 a 和 b，$M^l_{ab}\leqslant W[(a,b)_{\max}]$ 成立。

证明：假定 U 是超度量的且遵循夹层约束，但有一对对象 a 和 b 不满足上面的不等式，因此 $M^l_{ab}>W[(a,b)_{\max}]$。但根据定义 $(a,b)_{\max}$ 是 MST T 中从 a 到 b 通路中的最大权边。所以，对于 T 中从 a 到 b 的通路中的任何边 (x,y)，我们均能得到下面的不等式串：

$$M^h_{ab}\geqslant d^U_{ab}\geqslant M^l_{ab} > W((a,b)_{\max})\geqslant M^h_{xy}\geqslant d^U_{xy}\geqslant M^l_{xy}$$

现在加入边 (a,b)，并考虑 T 中创建的环［边 (a,b) 不能在 T 中，因为那样的话有 $(a,b)_{\max}=(a,b)$ 且 $M^l_{ab}>W((a,b)_{\max})=M^h_{ab}$，这显然不能发生］。在这个环中，由于上面中间不等式的限制，(a,b) 是惟一的最大权边，而这是一个矛盾。

定理的逆已在引理 6.4 中部分地证明了，剩下只需证明按算法计算的距离遵循夹层约束。我们先检查 M^l 的情况。考虑任意的顶点对 a、b，令 $e\in T$ 是使 a 和 b 在 U 中首先相连的边。我们有两种情况。假定 $e=(a,b)_{\max}$，则根据削权的定义 U 中 a 与 b 的距离是 $CW(e)\geqslant M^l_{ab}$。如果 $e\neq(a,b)_{\max}$，则 $CW[(a,b)_{\max}]\geqslant CW(e)$，且 $(a,b)_{\max}$ 是一条尚未被算法考虑过的边。但容易显示，当算法汇合 U 中的一对顶点 u、v 时，T 中从 u 到 v 的惟一通路上的所有的边（包括 e 在内）已被考虑过了，这是一个矛盾。

现在我们证明上限同样遵循这个定理。再次考虑任意的顶点对 a、b，令 $e\in T$ 是使 a 和 b 在 U 中首先相连的边。现在取顶点对 p、q 使得 $(p,q)_{\max}=e$ 且 $CW(e)=M^l_{pq}$。亦即，p、q 是用来确定 $CW(e)$ 的顶点对。p、q 对满足定理的条件，即 $M^l_{pq}\leqslant W[(p,q)_{\max}]$。但是 $W[(p,q)_{\max}]=M^h_e\leqslant M^h_{ab}$，后一个不等式成立，因为 T 是最小生成树且 e 位于 T 中从 a 到 b 的惟一通路中。 ■

我们以该定理结束用于距离矩阵的算法的讨论。我们注意到这里介绍的算法主要只有理论上的意义，部分原因是因为算法对输入数据作了太多的假设。这方面的研究者一般依赖各种启发式来重构树，习题 14 即为一例。 204

6.6 种系树之间的一致

在前面几节中我们介绍了重构种系发生的几种不同方法，在专业文献中还可发现更多的方法。在实际应用中，对同一组数据(或至少对同一对象集合)使用两种不同的方法产生不同的种系树(拓扑结构上的)十分常见。有时，甚至同一方法用于相同的数据(但数据排列不同)也会产生不同的树。在这些情况下，研究者应选择哪个树？这是一个困难的并引起广泛争议的问题。我们在本节通过介绍一个简单相关问题及其算法解来讨论这一问题。

问题如下：假设已知对于 n 个对象的同一集合有两个不同的有根树 T_1、T_2，每个内部节点恰有 2 个孩子(根除外，它可仅有 1 个)。我们想知道 T_1 和 T_2 是否一致，而一致的定义如下。

定义 6.6　每当通过缩减 T_r 中的某些边就可将树 T_r 转化为树 T_s 时，就称树 T_r 提炼(refine)树 T_s。如果有第三个树 T_3 能提炼树 T_1 和 T_2，T_1 和 T_2 就是一致的(agree)。

术语“树的相容性(tree compatibility)”也在文献中被使用。我们选择“一致”是为了避免与前一节所说的相容性概念混淆。直觉上，一致的概念意味着，不论什么信息被包含在 T_1 中但不被包含在 T_2 中，这些信息可被加到 T_2 中而不引发矛盾，反之亦然。彼此一致的树的例子见图 6.17 和图 6.18；提炼这两个树的树见图 6.19。图 6.20 是两个不一致的树的例子。这些例子是如此简单以至于一眼就可看出它们是否一致。当然，若树有几十或几百个叶子就会成为很大的问题。

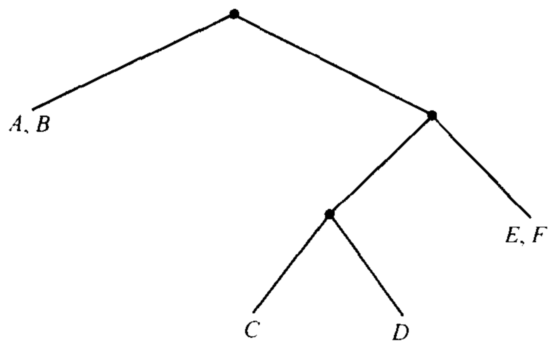


图 6.17　树 T_1

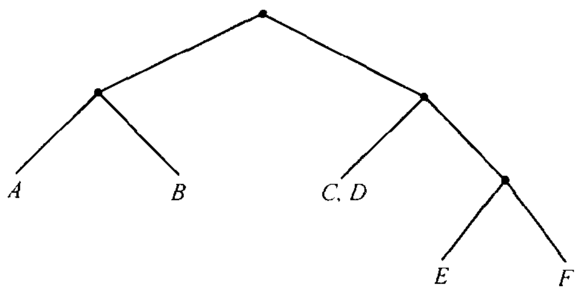


图 6.18　树 T_2 205

在前面的例子中，输入树有含几个对象的叶，这是一般的情况。作为第一步，我们首先处理每一个叶子仅含有一个对象或仅被一个对象标记的情况，在这种情况下，我们真正要做的是证实树 T_1 和 T_2 是否同构(isomorphic)。两个树 T_1 和 T_2 是同构的，当 T_1 和 T_2 的节点间存在一个一一对应，使得对每一对对应节点(u, v)，$u \in T_1$ 和 $v \in T_2$，u 下面叶子所包含的对象与 v 下面叶子所包含的对象相同。

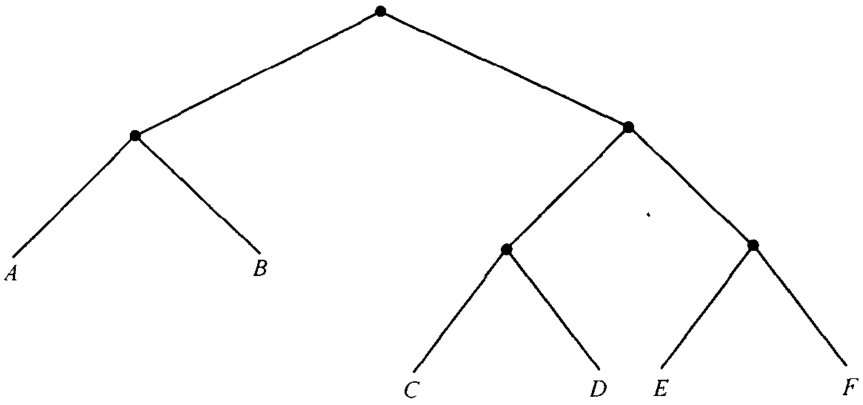


图 6.19　树 T_3,它提炼 T_1 和 T_2



206 图 6.20　两个不一致的树

尽管图的同构是个困难问题,二叉树同构可由简单的算法解决。基本思想是自底向上比较两个树,同时移去已经比较完的节点。一旦确定两对姐妹叶子(一对在 T_1 中,一对在 T_2 中)是同构的,就可移去它们,并把每一对的父节点标记为含有一个新的"人工"对象。在移去所有叶子后,它们的父节点转变为叶子,重复上述过程,直到处理完全部节点或发现树不同构。此算法依赖下面的数据结构和函数:集合 S,我们可以对其增加或移
207 去元素(对象)以及检测是否为空;函数 $object(u)$,它返回与节点 u 关联的对象;函数 $sib(u)$,它返回节点 u 的姐妹节点;函数 $parent(u)$,它返回节点 u 的父节点;节点的一个标记,告诉我们节点是否为叶子;数组 L_1,其下标是对象,它指示 T_1 中对象 i 出现的叶节点;以及用于 T_2 的类似的数组 L_2。算法显示在图 6.21 中。

该算法可看做对输入树 T_1 和 T_2 进行逐节点的仔细比较,故其正确性显而易见。运行时间为 $O(n)$,因为我们所做的就是遍历有 $O(n)$个节点的树常数次,且每一节点花费
208 常数时间。

我们现在考虑一般的情况,输入树的叶子可以包含几个对象。算法的思想与上面的同构检测算法类似,即逐个节点地仔细比较两个树。但不同的是,不从树中移去节点,而是根据 T_2 检查 T_1,并可能根据在 T_2 中发现的额外信息来修改 T_1。作为一个例子,考虑图6.17 中树 T_1 的含对象 A 和B 的叶节点。查看图6.18 中树 T_2 我们知道 A 和B 实际上应分为两个节点,因此据此修改 T_1。对 T_1 中的所有节点都可能做这种修改,然后同样地处理 T_2。在算法中,对象的数量储存在每一个子树中,这使我们可以很快地得知两个子树是否同构。如果 T_1 和 T_2 中对象的数量相同,则继续比较二者所存的对象。在对两个树进行了可能的修改后,我们对(可能)修改后的 T_1 和(可能)修改后的 T_2 进行同构检测,这个检测本质上与图 6.21 中的一样。更详细的算法见图 6.22。

```
Algorithm Binary Tree Isomorphism
  input: Trees T1 and T2
  output: TRUE if T1 and T2 are isomorphic, FALSE otherwise
  // Initializations
  k ← n  // k counts the number of objects plus "new" objects
  By appropriate depth-first searches in tree T1 do
      if node u is a leaf then
          L1[object(u)] ← u
          if sib(u) is a leaf then
              add(S, u)
  By appropriate depth-first searches in tree T2 do
      if node u is a leaf then
          L2[object(u)] ← u
  // Main loop
  while S ≠ ∅ do
      u ← remove(S)
      j ← object(u)
      v ← L2[j]
      y ← sib(u)
      w ← sib(v)
      if w is not a leaf or object(w) ≠ object(y) then
          return FALSE
      p1 ← parent(u)
      p2 ← parent(v)
      Remove nodes u, v, w, y from respective trees
      Mark p1 and p2 as leaves
      k ← k + 1
      L1[k] ← p1
      L2[k] ← p2
      // p1 and p2 now contain "new object" k
      if sib(p1) is leaf then
          add(S, p1)
  // end while
  return TRUE
```

图 6.21 有根二叉树同构测试

这种算法的运行时间分析如下。初始化时，花费 $O(n)$时间来遍历树常数次。在主循环中，注意到一旦节点被宣布失活，则在以后的算法中均保持失活，而每一个活跃节点最多被检查一次。检查节点是否含同样的对象需要遍历一次每个树的每一部分。至此我们花费的时间为 $O(n)$。最后还有同构检测，我们必须修改先前介绍的同构算法，因为在输入树中叶子可以包含多个对象。首先，函数 *object*(u)必须返回与叶子 u 相关的所有对象(见图 6.21 的起始部分)；然后在算法的主循环中，检测两个节点是否被同一对象所标记的算法需要修改，以检查集合的相等性。我们把实现这些修改并保证检查耗费的总时间为 $O(n)$留作习题 18，这个算法总的运行时间因此是 $O(n)$。

至此，我们结束了对种系树一致问题的简短介绍。存在着更复杂的问题形式，该领域工作的其他文献见本章的文献提要。 209

```
Algorithm Rooted Binary Tree Agreement
  input: Trees T1 and T2
  output: TRUE if T1 and T2 agree, FALSE otherwise
  By an appropriate depth-first search in tree T1 do
    if node u is leaf then
      for every object i stored in leaf u do
        L1[i] ← u
        K1[i] ← number of objects stored in leaf u
    else J1[u] ← number of objects in this subtree
  By an appropriate depth-first search in tree T2 do
    if node u is leaf then
      for every object i stored in leaf u do
        L2[i] ← u
        K2[i] ← number of objects stored in leaf u
    else J2[u] ← number of objects in this subtree
  for each object i do
    active[i] ← TRUE
  // we now examine tree T2 with respect to T1 and possibly modify T1
  for each object i do
    if active[i] and K1[i] > K2[i] then
      // T2 contains more information about object i
      w ← L2[i]
      repeat
        w ← parent(w)   // we go up in the tree
      until J2[w] ⩾ K1[i]
      // We have reached the root w of a subtree of T2 that has
      // at least as many objects in it as the leaf of i in T1
      if J2[w] ≠ K1[i] then return FALSE
      Traverse subtree of T2 rooted at w checking that
        every object of L1[i] is in a leaf of this subtree
      if this test fails then return FALSE
      Replace node L1[i] in T1 with subtree of T2 rooted at w
      for each object j stored at L1[i] do
        Update L1[j]
        active[j] ← FALSE
  // End of outermost for
  // here we must repeat what was done above, but this time examining T1
  // with respect to T2 and possibly modifying T2 in the process
  // (similar code omitted)
  Run algorithm Binary Tree Isomorphism on T1 and T2
  return TRUE if trees are isomorphic and FALSE otherwise.
```

图 6.22 检测两个有根二叉树一致的算法

小　结

种系树对帮助理解随时间进化的对象(特别是对分子序列)之间的关系是很重要的工具。构造种系树需要将对象进行相互之间的比较,对此有两条基本标准。一个标准是利用离散性状,我们选择一组性状,对每一个对象列出该对象的每一个性状的状态。将全部

的列表集合起来形成一个性状状态矩阵,从此矩阵即可构造种系树。其基本思想是,共享状态愈多的对象在种系树中愈接近。另一个标准是对象之间的距离,而距离的概念取决于对象之间的分组。类似地,其基本思想是,距离愈近的对象在种系树中的位置也越近。

如果所比较的数据在一个性状状态矩阵中,且如果我们对期望的树施加某些限制,树的重构就变为完全种系树问题。根据性状,这个问题可以是 NP-完全的。对于某些存在高效算法的特殊例子我们提供了算法,一个算法用于二值状态的情形,另一个算法用于仅有两个性状的情形。在后一例,我们看到图理论大大有助于算法设计,并是更一般算法的基础。

实际的数据几乎不支持完全种系树,因此必须放松对所期望的树的限制条件,这导致类似于最俭约树或最大相容集这样的优化问题。显然,这些问题是 NP-难题,意味着这些问题的算法要么执行穷尽搜索要么设法逼近最优解。

当对象间通过距离进行比较时,我们可能要求观察的距离与重构树上测得的距离相等,这是可加树问题,而我们介绍了一个简单算法来解决该问题。但同样,实际的数据很少支持可加树。一个解决的办法是构造一个距离遵循上下限的树。当我们施加额外的限制时,即树是超度量的(所有的叶—根路径必须有同样的长度),就有可能设计出有效的算法。我们描述了这种算法。

因为存在很多构造种系树的方法,因此需要比较基于同一个对象集的不同的树。我们介绍了一个简单的算法根据两个树来确定一个树,在输出树中使用了两个输入树的信息。只要两个输入树在某种意义上精确地一致,这个算法就得出一个一致树。

习　　题

1. 种系树有很多应用,其中两个应用是对语言和古代文稿进行分类。试图把有序和无序性状的概念和距离的概念用于这两个应用中。 210
2. 证明带标记的无根二叉树的数量是 $\prod_{i=3}^{n}(2i-5)$。(提示:从找出这样的树含多少内节点和内边开始,然后应用归纳法,注意,当往树中增加一个对象时会发生什么,该对象使一条边分叉。)
3. 提出一个有根种系树的线性表示,这种表示可用于一个种系树计算机程序的输出。
4. 图 6.5 的算法假定所有的列互不相同,如何修改算法以适用于有相同列的情况?
5. 6.5 节中分两阶段介绍二值状态的算法,如何将二者整合成一个阶段?
6. 注意引理 6.1 与 5.3 节中连续 1 问题算法条件之间的相似性。证明包含一个完全种系树的二值性状矩阵对列有 C1P 性质。利用这种矩阵的特殊性质,你是否能设计一个算法找出一个能产生连续 1 性质的行置换?
7. 利用书中陈述的事实,证明 6.2 节末尾描述的转换方法能把无序二值性状转化为有序二值性状。
8. 假设我们有如下的性状状态矩阵:

	c_1	c_2
A	0	1
B	1	1
C	1	0

根据二值性状的结果(6.2 节),这个输入不包含完全种系树,因为 c_1 和 c_2 是不相容的。但是,根据两个性状的结果(6.3 节),这个输入有一个完全种系树,因为 SIG 是无环的。哪一个解释正确?

9. 如果一个基于性状状态的种系树是完全的,我们应该能够给每一条边分配一个性状状态转换。给图 6.8 的树的每一条边分配这样的转换。

* * 10. 当每一个性状的状态的数量最多是 3 时,设计一个完全种系树问题的算法。

* * 11. 当性状的数量最多是 3 时,设计一个完全种系树问题的算法。

* 12. 当性状的数量最多是 3 且每一个性状的状态的数量最多是 3 时,设计一个完全种系树问题的算法。

13. 假定我们有一个算法能够找出图的最大圈,且我们想用它解决一般性状状态相容问题。一般地性状状态未必是二值的。定理 6.5 的证明说明了当性状状态是二值的时如何解决这个问题。对于一般的性状状态情况,如何修改这个算法?

211 14. 一个众所周知的应用距离数据构造种系树的启发式方法如下。选择彼此最紧密的对象对 A 与 B,令它们为一个新对象 O 的孩子,每边设置长为 $d(A,B)/2$;现在从距离矩阵中移去 A 和 B,并添加 O,令 O 与每一个其他节点 i 之间的距离是 $[d(A,i)+d(B,i)]/2$;再找出彼此最紧密的对象对,重复上面的过程直到所有的对象都在树中。(当用一个内节点连接两个对象时,我们必须考虑被连接对象边的边长)。对表 6.5 中的矩阵应用此启发式,是否可发现一个可加树?

15. 在介绍重构可加树的算法时,我们假设输入矩阵 M 是可加的。假设我们并不知道 M 是否是可加的,这时该如何办?

* * 16. 假设在可加树问题中,我们知道内节点有受限的度。证明在这种情况下有可能获得运行时间为 $O(n\log n)$的算法。

17. 证明超度量距离具有下面的特征:对任何对象 i、j 和 k,我们有 $M_{ij}\leqslant\max\{M_{ik},M_{jk}\}$。

18. 修改图 6.21 中的树同构算法,以解释叶可包含多于一个对象的可能性。

19. 证明引理 6.3。

文 献 提 要

本章开始已提及,相对于这个领域的广泛性,我们对种系发生树算法的探讨是狭窄的。特别是,我们没有涉及基于统计分析的种系树构造算法,这种算法即是广为所知的最大似然方法。现在,我们为对这个问题有兴趣的读者提供一些文献。

对于以分子数据为基础的种系树重构,对更多细节起点性的介绍可见 Swofford 和 Olsen 的出色综述[182]。另一篇同样很好但略为陈旧的文献是 Felsenstein 的[60]。Nei 的关于分子演绎的书在第 11 章给出了一个有用的综述[147]。Penny、Hendy 和 Steel 做了简

短的回顾[159]。

在 6.1 节提及的无序性状完全种系树问题，已被 Bodlaender、Fellows 和 Warnow 证明是 NP-完全的[25]，Steel 亦独立地证明了这个问题[179]。6.2 节的算法归功于 Gusfield[82]，Waterman 作了简化[199]。McMorris 展示了如何把无序二值性状转化为有向二值性状[132]。

正如本章所述，当性状的数量固定时，完全种系树问题可在多项式时间内解决，这个发现归功于 Agarwala 和 Fernández-Baca[4]，他们的算法的运行时间为 $O(2^{3r}(nm^3+m^4))$。也有人提出了针对 r 的特定值的算法。例如，当 $r=3$ 时存在两个算法：Dress 和 Steel 所提议的运行时间为 $O(nm^2)$ 的算法[52]，以及 Kanna 和 Warnow 所提议的运行时间为 $O(n^2m)$ 的算法[107]。Kannan 和 Warnow 提供的算法在 $r=4$ 时依然成立。这是一个非常重要的算法，因为它可用于无空隙的 DNA 序列比对。在这种情况下，我们可把比对中的每一位置看做一个性状，该性状可处于 4 种状态(A、C、G、T)之一。 212

Agarwala、Fermández-Baca 和 Slutzki[5]，以及 Meidanis 和 Munuera[135]介绍了一种用于二叉种系树的算法，输入数据是每个对象的性状表，而不是矩阵。

6.3 节的内容基于 Warnow 的论文[194]。定理 6.1 由 Buneman 给出[29]，表 6.3 和图 6.7 也选自同一篇文章。McMorris、Warnow 和 Wimer 描述了一个算法[133]，适用于固定数量的性状，运行时间为 $O(r^{m+1}m^{m+1}+nm^2)$。有三篇文章分别描述了当 $m=3$ 时的算法，作者分别是 Bodlaender 和 Kloks[26]、Kannan 和 Warnow[106]，以及 Idury 和 Schäffer[101]。

简约标准导致 NP-完全问题的证明出现在 Day、Johnson 和 Sankoff 的论文中[43]。在这篇论文中，他们用到了把种系树与 Steiner 树问题联系起来的重要结果，这个结果早先由 Foulds 和 Graham 得到[64]。前面提到的 Swofford 和 Olsen 的综述为俭约问题提供了更多的线索。Swofford 和 Maddison 提供了一个简约算法用于祖先性状状态赋值[181]，该算法基于另一个称做 Wanger 俭约标准的标准。在 6.4 节中的相容性标准的 NP-完全性证明是由 Day 和 Sankoff 作出的[44]。

出现在 6.5.1 中的 4 点条件由 Buneman 证明[202]，也由 Dobson 独立地证明[49]。6.5.1 介绍的可加树构造算法来自 Waterman、Smith、Singh 和 Beyer[202]。当内节点有受限的度时，Culberson 和 Rudnicki[40]以及 Hein[90]介绍了更快的算法。在各种度量条件下获得能最佳地逼近可加树的树被 Day 证明是 NP-完全问题[42]。夹层超度量树的算法由 Farach、Kannan 和 Warnow 给出[56]，在这篇论文中他们进一步说明了树必须是超度量的，因为找到一个能满足夹层约束条件的可加树是一个 NP-完全问题。另外他们还介绍了其他与可加树和超度量树有关的复杂性结果。Agarwala 和其他作者使用超度量树的结果提出了一个带性能保证的用于可加树问题的近似算法[3]。他们进一步证明，获得比某个阈值更好的近似值是 NP-难题。

在 6.6 节介绍的算法归功于 Gusfield[82]。当树不一定是二叉的时，Warnow 介绍了一个更一般的算法[195]。她还说明，对于无序性状完全种系树问题，这个算法可在一个分枝界限算法中被使用。由 Amir 和 Keselman[14]得出的关于树的一致的最近结果表明，当 $k=3$ 时，获得一个与 k 个输入树一致的子树而同时将子树含有的对象数量最大化是一个 NP-难题。对于其中一个树具有受限的度的情况，他们介绍了一个多项式时间的算法，这

个结果后来被 Farach、Przytycka 和 Thorup 改进[57]。

当所研究的对象是分子序列时,常常用序列多重比对的结果作为输入数据进行种系树的重构。另一方面,种系树对于序列比对也是很有用的。这种情况导致很多研究人员
213 提出同时解决这两项任务的算法。这个问题的一个版本叫做“树比对”,3.4.3 简短地说明了这种情况。这个课题的某些线索在第 3 章的文献提要中介绍,其他关于多序列比对和种系树重构组合问题的启发式解法的文献见 Hein[89,91,92]以及 Vingron 和 von Haeseler[190]。

两个常用的种系树重构软件包是 Felsenstein 提供的 PHYLIP(phylogeny inference package)[61]和 Swofford 提供的 PAUP(phylogenetic analysis using parsimony)[182],二者均
214 提供了许多建树的方法。

第 7 章　基因组重排

在前面的章节中我们看到如何利用基因在碱基序列水平上的差异来推断相关物种间的进化关系。但也有一些例子表明，两个或多个物种间基因的比较所得到的信息要少于对更大基因组单元甚至是整个基因组的比较所得到的信息。更大单元的比较在一个更高的层次进行，它不是关注于序列，而是比较若干相关基因的位置并试图确定能把一个基因组转化为另一个基因组的基因重排操作。人们已发展了丰富的数学理论来模拟这种情形，本章仅对这些理论作导论性的介绍。

7.1　生物学背景

当比较不同物种的全基因组时，需要计算物种间的“距离”，而我们决定采用一组不同 215
于第 3 章所介绍的基本操作。我们此时对包括替换、缺失和插入在内的点突变不再关注，而对更大的突变感兴趣。染色体的片段可进行几种不同方式的交换，这影响了很大比例的染色体。一个染色体小片可以被移动或复制到另一个位置，甚至从一条染色体离开而并入另一条染色体。这样的移动称做**基因组重排**(genome rearrangement)。

在一些例子中，例如，当研究叶绿体和线粒体基因组时，仅有的一种重组事件，即反转，几乎是影响基因组进化的惟一方式。在这种情况下，用把一个基因组转换为另一个基因组所需的反转操作的数量来度量两个基因组间的进化距离是有道理的。

作为一个例子，考虑紫花苜蓿和豌豆的叶绿体基因组，它们是两个相关的植物物种，
如图 7.1 所示。图中每个标记的箭头表明一个块(block)，它是可能包含多个基因的一段 216
基因组，整段基因组作为一个转录单元。箭头所指表明块具有朝向，其理由是转录发生于特定的链上，这与 1.4.2 中的解释一样。不同基因组的两个块如果同源，即如果它们含有同样的基因，则有相同的数字标记。最后需注意的一点是，在一对基因组中，基因间的距离并不总是相同的。

有向块的反转操作可以这样定义：对块的邻接片段进行操作，反转相关块的序，并翻转箭头方向。面对这个定义，我们感兴趣的是下面的问题：导致一个基因组变为另一个基因组的最小反转操作数量是多少？一种可能的解由图 7.2 给出，第一行表示紫花苜蓿的基因组，最后一行表示豌豆的基因组，除第一行外，其他行都是将下划线标记的块反转而

得到。可是，我们如何知道不存在更短的操作序列可完成所需的变换？我们希望有多项式时间的算法可给出该问题的答案，而这正是本章的论题。

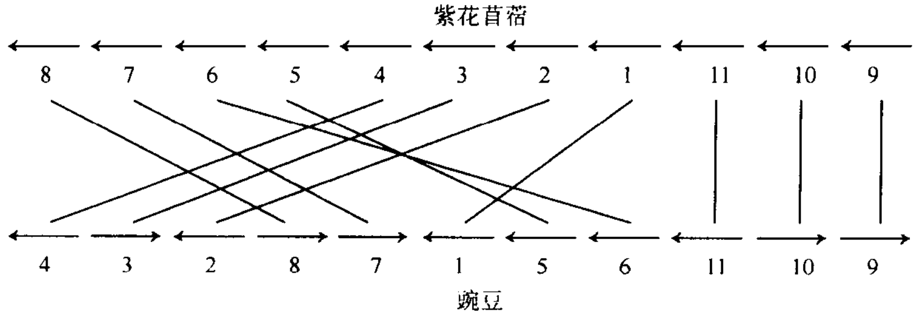


图 7.1　紫花苜蓿和豌豆叶绿体基因组的关系(引自文献[150])

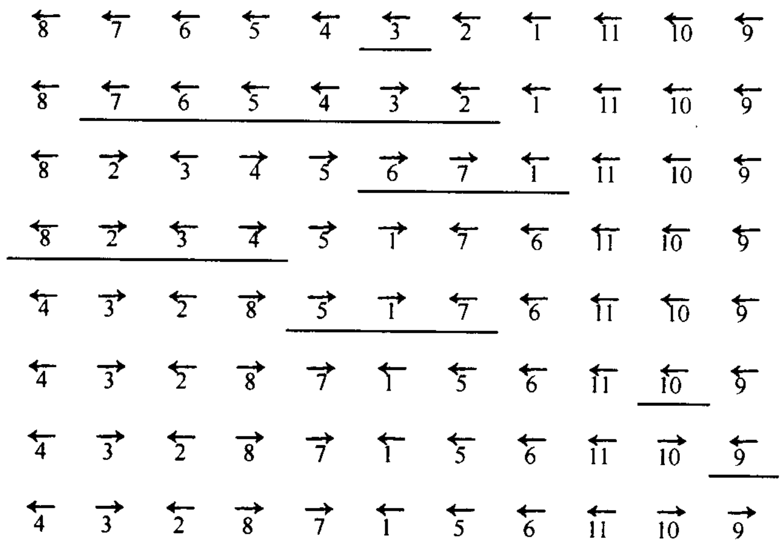


图 7.2　图 7.1 问题的一个解

217　同本书其他部分处理最小化问题一样，我们想获得最短可能反转操作序列的理由也是简约假设(parsimony assumption)。我们假定自然界总能找到所需变化最小的途径，因此如果我们要知道紫花苜蓿如何可转变为豌豆，我们就需找出一条完成这种转换的最小反转操作序列。

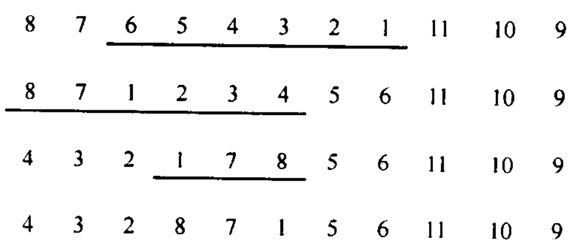


图 7.3　同一问题的另一个解，块是无向的

可以构想与之类似的无向块的重排问题。有时我们所获得的有关基因的信息虽未准确到可知道它们的相对朝向，但知道它们的相对次序。在这种情况下，仅需寻找可将一种块次序转换为另一种块次序的最短反转操作序列即可，不必考虑其朝向。例如，假定我们不知道图 7.1 中块的朝向，在这种情况下，3 个反转操作就足以解决问题，见图 7.3。

下一节讨论这一问题的两种版本，即所谓的**反转排序**(sorting by reversal)。有趣的是，尽管对有向块存在多项式时间的发现最小反转序列的算法，对无向块则不行，它反而是一个 NP-难题。

7.2　有向块

我们以一个例子开始来研究有向块问题的解法。假定我们把两个基因组分成图 7.4 所示的同源块，每一块表示两个基因组中高度保守的连续片段。在这个例子中，3 次反转可以把一个基因组转换为另一个基因组，且这是最少的步骤。当然还存在其他的反转过程，但均需至少 3 次的反转。 218

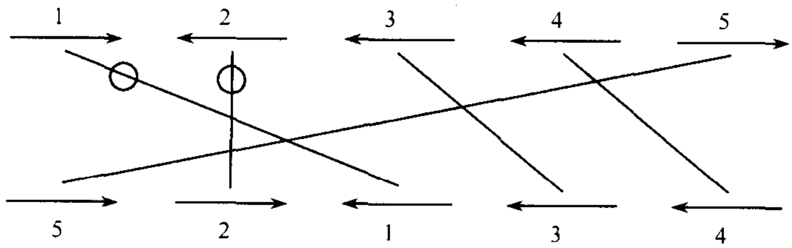


图 7.4　具有同源块的两个基因组，箭头表示每一基因组的块，同源块以直线连接，直线附加小圆圈表示对应块在两基因组中的方向不同

图 7.5 显示了一个可能的解，每一块上的标记显示了反转的操作，标记上的箭头指明了同源块的方向。因为标记可以是任意的，我们使用数字 1 到 n(本例中 $n=5$)按自然顺序来标记上面的基因组。用同样的标记，下面的基因组中的标记亦被惟一地确定。每一块与其同源块具有相同的数字，而箭头反映了其实际朝向。在该图中，反转以所涉及块的下划线指明。这样，初始的排列(permutation)被一步一步地转变为最终的排列。

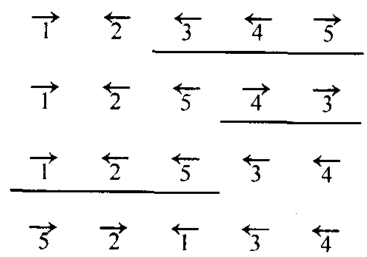


图 7.5　上图问题的一个解，其中每一行通过一个反转操作转换自前一行，反转涉及的块以下划线标记

如何知道不存在更少的反转步骤可完成所需转换？在这个特定例子中，有简单的证据可证明少于 3 个反转是不行的，这个证据依赖于**断点**(breakpoint)的概念。一个断点是初始排列中连续标记之间的一个点，该点必须为至少一个反转所分隔以便达到最终的排列。换句话说，在初始排列中两个连续的标记在目标排列中是不连续的，或者即便仍连续但朝向肯定不同。另外，如果初始排列的第一个标记在目标排列中不再是同一朝向的第一个标记，则在第一个标记前存在断点。图 7.6 显示了图 7.4 例子初始排列的断点，每一断点以"•"表示。例如，在标记 $\overleftarrow{2}$和 $\overleftarrow{3}$之间有一个断点，因为在目标排列中 $\overleftarrow{2}$和 $\overleftarrow{3}$不连续。若没有反转隔开这两个标记，则直至最后二者仍将在 219



图 7.6　每一个"•"是一个断点

一起，形成错误的次序。相比之下，$\vec{1}$和 $\overleftarrow{2}$之间无断点，因为这两个块正好连续出现在目标序列中。图 7.4 的初始排列共有 4 个断点。

一个反转最多可去掉两个断点，因为它恰在两个位置分切排列。目标排列有 0 个断点，且是具有此特性的惟一排列。所以在本例中我们至少需要两个反转以确定 4 个断点。但仔细观察发现，不存在单一的反转可移去初始排列中的 2 个断点。实际上，要移去第一个断点，即 $\vec{1}$前面的断点，我们必须将 $\vec{5}$移到第一个位置，这甚至还不够。惟一能移动 $\vec{1}$的反转亦改变 $\vec{1}$的方向，这样产生的排列在第一个位置前仍有一个断点，标记处在正确的位置，但朝向是错的。

类似的理由使我们确信亦没有办法仅用一次反转移走最后一个断点。因此，一次反转移走两个断点只可能发生在中间断点上了。然而，影响这两个中间断点的惟一的反转并不能移走任何一个断点！因此，我们现在知道第 1 次反转最多只能使断点的数量减少 1 个。然后，至少还剩下 3 个断点，需要另外两次反转，因此总共至少需要 3 次反转，这表明前面所示的解是优化的。

7.2.1　若干定义

对基因组重排的有关定义必须十分小心，并使用一致的标记系统。让我们首先定义什么是有向排列。给定一个有限的标记集$\mathcal{L}$，有向标记来自集合$\mathcal{L}^0$，这里

$$\mathcal{L}^0 = \bigcup_{a \in \mathcal{L}} \{\overleftarrow{a}, \vec{a}\}$$

集合$\mathcal{L}^0$ 的元素个数是$\mathcal{L}$的两倍。对于 $x \in \mathcal{L}^0$，以$|x|$表示移去 x 的箭头后的$\mathcal{L}$中的元素。因此，对任何 $a \in \mathcal{L}$有$|\overleftarrow{a}| = |\vec{a}| = a$。令 n 为$\mathcal{L}$中元素的数量，$\mathcal{L}$上的一个有向排列是一个映射 $\alpha:[1\cdots n] \mapsto \mathcal{L}^0$，使得对任何标记 $a \in \mathcal{L}$，恰存在一个 $i \in [1\cdots n]$满足$|\alpha(i)| = a$。不严谨地讲，一个排列为每一个数标 i 从$\mathcal{L}^0$ 中“拾取”一个有向标记 $\overleftarrow{a}$或 $\vec{a}$。但是，如果 $\overleftarrow{a}$被某一数标选取了，则 $\vec{a}$便不能被任何其他数标选取，反之亦然。这样 $n = |\mathcal{L}|$确保了对每一个 $a \in \mathcal{L}$，$\overleftarrow{a}$或 $\vec{a}$只能被拾取一次。

我们把排列 α 表示为一列元素$\alpha(1), \alpha(2), \ldots, \alpha(n)$。同上节一样，通常以空格分隔
220 元素序列，但有时为避免歧义我们使用逗号或括号。在大多数情况下，集合$\mathcal{L}$将恰是前 n 个正整数 $1,2,\ldots,n$。因此，如果 $n=6$，则下面就是一个在$\mathcal{L} = \{1,2,\cdots,6\}$上的排列：

$$\alpha = (\overleftarrow{2}, \overleftarrow{3}, \vec{1}, \vec{6}, \overleftarrow{5}, \overleftarrow{4}) \tag{7.1}$$

所以在上面的例子中，$\alpha(1) = \overleftarrow{2}$，$\alpha(2) = \overleftarrow{3}$，依此类推。恒等排列(identity permutation)是指排列 I，对所有的介于 1 到 n 之间的 i 均有 $I(i) = \vec{i}$。

反转(reversal)是把一个排列转化为另一个排列的操作，把连续部分的次序反转并同时翻转相关元素的方向符号。令 i 和 j 为两个满足 $1 \leqslant i, j \leqslant n$ 的数标，通过 $\alpha(j)$影响元素 $\alpha(i)$的反转用$[i,j]$表示。反转实际上是$\{1,2,\ldots,n\}$到其自身的一个映射，它可与一个有向排列 α 相复合以给出另一个从$\{1,2,\ldots,n\}$到$\mathcal{L}^0$ 的映射(即另一个排列)。映射 α 和 ρ 的复合记做 $\alpha\rho$，它是如下定义的映射：

$$(\alpha\rho)(k) = \alpha(\rho(k))$$

例如,对等式(7.1)的排列 α 进行[2,4]反转,产生

$$\alpha[2,4] = (\overleftarrow{2},\overleftarrow{6},\overleftarrow{1},\overrightarrow{3},\overleftarrow{5},\overleftarrow{4})$$

通常,一个反转 $\rho=[i,j]$把一个排列 α 改变成$\alpha\rho$,它定义为:

$$\alpha[i,j](k) = \begin{cases} \overline{\alpha(i+j-k)} & \text{如果} \quad i \leqslant k \leqslant j \\ \alpha(k) & \text{否则} \end{cases}$$

这里 $\bar{l}$ 是标记 l 方向反转后的结果。当有 n 个块时,存在 $n(n+1)/2$ 个反转,其中包括仅改变 $\alpha(i)$箭头方向的单式反转$[i,i]$。

我们在本节中的主要兴趣是反转排序问题。假定在同一标记集合$\mathcal{L}$上有两个有向排列 α 和β,我们要寻找能够把 α 转化为β 的最小反转数。换言之,使用刚才介绍的符号,我们寻找一系列反转 $\rho_1,\rho_2,\dots,\rho_t$,数量应尽可能少,使得:

$$\alpha\rho_1\rho_2\dots\rho_t = \beta$$

数字 t 称为α 的相对于β 的反转距离,记做 $d_\beta(\alpha)$。有向排列 α 称为初始排列,而 β 称为目标排列,该过程叫做相对于 β 排序 α。

注意,尽管对于给定的 α 和β, $t=d_\beta(\alpha)$是固定的,但实际的对 α 进行排序的反转序列一般并不惟一。通常,我们感兴趣的是发现一个可向 β“推进”的反转 ρ,这样的反转叫做 α 关于β 的排序反转,它具有下述特性:

$$d_\beta(\alpha\rho) < d_\beta(\alpha)$$

等价于

$$d_\beta(\alpha\rho) = d_\beta(\alpha) - 1$$ 221

因为 α 和$\alpha\rho$ 的距离不能超过一次反转。

观察到反转距离具有对称性非常重要,也就是 $d_\beta(\alpha)=d_\alpha(\beta)$,这起源于这样的事实,即任何反转如果施行两次则不产生实际净效果。换言之,对于任何反转 ρ, $\rho\rho$ 无任何效果。所以,如果

$$\alpha\rho_1\rho_2\dots\rho_t = \beta$$

则用 $\rho_t,\rho_{t-1},\dots,\rho_1$ 依次复合等式两边我们得到:

$$\alpha = \beta\rho_t\rho_{t-1}\dots\rho_1$$

我们可得出结论,即同样的反转可使 β 转化为α,但须以相反的次序应用这些反转。因此,关于 α 的β 排序反转未必与关于β 的α 排序反转相同。

若目标排列 β 在上下文中是清楚的,我们就将 $d_\beta(\alpha)$省略为 $d(\alpha)$。类似地,在同样情况下,当讨论排序反转、断点等内容时亦省略 β。

7.2.2　断点

我们给出断点的正式概念。给定排列 α,定义断点需做的第一件事是考虑 α 的扩展形式,在第一个标记 $\alpha(1)$之前和最后一个标记 $\alpha(n)$之后分别加入人为标记 L(左)和 R(右)得到 α 的扩展形式。式(7.1)的排列 α 的扩展形式是:

$$(L,\overleftarrow{2},\overleftarrow{3},\overrightarrow{1},\overrightarrow{6},\overleftarrow{5},\overleftarrow{4},R) \tag{7.2}$$

α 关于β 的一个断点是指$\mathcal{L}^0$ 的一对元素 x、y, xy 出现在α 的扩展形式中,但 xy 和

$\bar{y}\,\bar{x}$ 均不出现在β的扩展形式中。例如，如果 α 是(7.2)式的排列，β 是恒等排列，则断点是：$L\ \overleftarrow{2}$，$\overleftarrow{2}\overleftarrow{3}$，$\overleftarrow{3}\overrightarrow{1}$，$\overrightarrow{1}\overrightarrow{6}$，$\overrightarrow{6}\overleftarrow{5}$和 $\overleftarrow{4}R$。$\overleftarrow{5}\overleftarrow{4}$不是断点，因为其反向对 $\overrightarrow{4}\overrightarrow{5}$出现在 β 中。

排列 α 的断点数量记做$b_\beta(\alpha)$，或当上下文中 β 明确时简记为$b(\alpha)$。在这个例子中，$b(\alpha)=6$。

注意，人为标记 R 或L 始终不移走，故仅当 α 和β 的第一个标记不同或标记相同但朝向不同时，L 才与断点有关；类似的情况亦适用于 R。

=========== 一 个 下 界 ===========

令 β 固定，如先前所述，一个反转最多可从一个排列中移去两个断点。使用记号表示为：

$$b(\alpha)-b(\alpha\rho)\leqslant 2 \tag{7.3}$$

α 是任何排列，ρ 是任何反转，它立即使我们得到如下反转距离的一个下界。令 $\rho_1,\rho_2,\ldots,$
222 ρ_t 为将 α 转变为 β 的一系列反转(未必是优化的)，则

$$\alpha\rho_1\rho_2\ldots\rho_t=\beta$$

因此

$$b(\alpha\rho_1\rho_2\ldots\rho_t)=b(\beta)=0 \tag{7.4}$$

注意 β 关于其自身无断点。可是，根据式(7.3)

$$b(\alpha)-b(\alpha\rho_1)\leqslant 2$$
$$b(\alpha\rho_1)-b(\alpha\rho_1\rho_2)\leqslant 2$$
$$\vdots$$
$$b(\alpha\rho_1\ldots\rho_{t-1})-b(\alpha\rho_1\ldots\rho_t)\leqslant 2$$

相加这些不等式，并考虑式(7.4)，我们得到：

$$b(\alpha)\leqslant 2t$$

该等式对 α 的任何排序都成立。现在将该结果用于一个最短反转系列，则 $t=d(\alpha)$，且有

$$\frac{b(\alpha)}{2}\leqslant d(\alpha)$$

这是 $d(\alpha)$的一个下界，它是一个确定 $d(\alpha)$为真的最小值。如果对一个特定的排列 α，一旦我们发现一个大小为下界的反转序列，我们便知道已找到了一个解。然而，这个下界并不总是可得到的。例如，在 7.2 节中我们看到一对排列 α 和β，$b_\beta(\alpha)=4$，因此下界为 2，但 $d_\beta(\alpha)=3$。

一个令人惊异的事实是，存在着可移去两个断点但却不是排序反转的反转。对于排序 α 而言，这些反转使我们误入歧途，见习题 5。

7.2.3 现实与期望图式

下界 $d(\alpha)\geqslant b(\alpha)/2$ 并不紧致，对于很大一部分排列它远不是一个等式。在这一部分我们导出一个更好的下界，基于一种称为一个排列相对于另一个排列的现实—期望图式的结构(reality-desire diagram)，下面将定义该结构。

为获得该结构的直觉印象，首先让我们确定是什么使得一个反转从排列 α 中移走两个断点。一个一般的情况描述如下，其中反转以两条竖线表示：

$$\cdots x \mid y \cdots z \mid w \cdots$$

在标示的反转被执行后，该序列变为：

$$\cdots x\,\bar{z} \cdots \overline{y} w \cdots$$

这里，$\bar{l}$ 表示反方向的标志 l。假定断点 xy 被移走，则要么 $x\bar{z}$ 在 β 中，要么 $z\bar{x}$ 在 β 中。 223
然而 xy 是一个断点，所以 xy 和 $\overline{yx}$ 都不出现在 β 中，尽管 xy 出现在 α 中。

当有一断点时，它是由 α 中两个相邻标记产生的，但这两个标记并不“愿意”相邻，它们“企图”与其他标记相邻。这样，我们观察到了现实与期望之间的偏离，现实是某个标记在 α 中的相邻标记，愿望是该标记在 β 中的相邻标记。实际上，标记是有方向的，因此它们对每个端点都有一个特定的期望。一个有向标记可看做是具有正负极标记的电池，如图 7.7 所示。

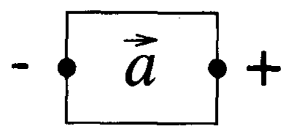


图 7.7 类似电池的有向标记，正端总在箭头顶上，而负端在箭头尾

然后，我们分析一下上述例子中特定的现实与期望状态。我们用一个盒子表示一个标记。另外，我们用“现实”线连接当前排列中相接触标记的端点，用“期望”线连接目标排列中相接触标记的端点，其结果如图 7.8 所示。注意，断点的移去导致一条连接相同末端(terminal)的期望线，即经该次反转后期望变为现实。在这个特定的例子中我们不知道哪个末端是正的哪个末端是负的，但这并不是最关切的问题。

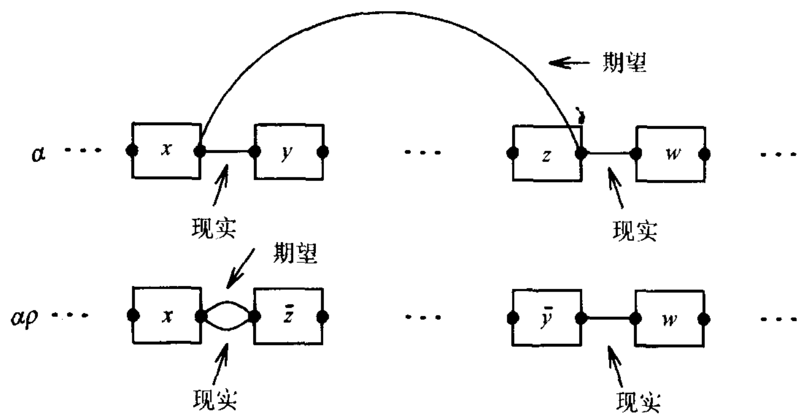


图 7.8 断点移除过程中的现实与期望线

上述考虑促使我们构建如下结构的排列 α 关于 β 的**现实—期望图式**(reality and desire digram)，记为 $RD_\beta(\alpha)$。每个标记用两个结点表示，以便我们能跟踪标记的朝向；对每个标记 $l \in \mathcal{L}$，我们在 $RD(\alpha)$中包含两个结点 $\dot{\vec{l}}$和 $\vec{l}$，结点 $\dot{\vec{l}}$代表箭尾而 $\vec{l}$代表箭头；另外增加两个额外结点 L 和 R 以记录末端标记(end label)的现实与期望线。对所有的排列都如此标记。 224

图式 $RD(\alpha)$有两类线(或边)：现实边和期望边。现实边连接排列 α 中连续标记的末端，且 L 和 R 分别与第一个和最后一个末端连接，期望边则根据排列 β 连接标记末端。最后，所有的末端结点排成一圈，L 和 R 占据顶上两个位置，L 在 R 左侧，其他结点则遵

循 α 的反时针序。图 7.9 显示了排列 $\overrightarrow{3}\overleftarrow{2}\overleftarrow{1}\overrightarrow{4}\overleftarrow{5}$ 相对于恒等排列的现实与期望图式的逐步构造过程，图 7.10 显示了排列成圈的该图式，现实边居于圆周上，期望边是弦。注意，有时在同一对顶点间存在两条并行边，一条是现实边另一条是期望边，这仅发生在不是断点的结点间，此时，现实边等价于期望边。另一方面，不存在自反环(self-loop)。

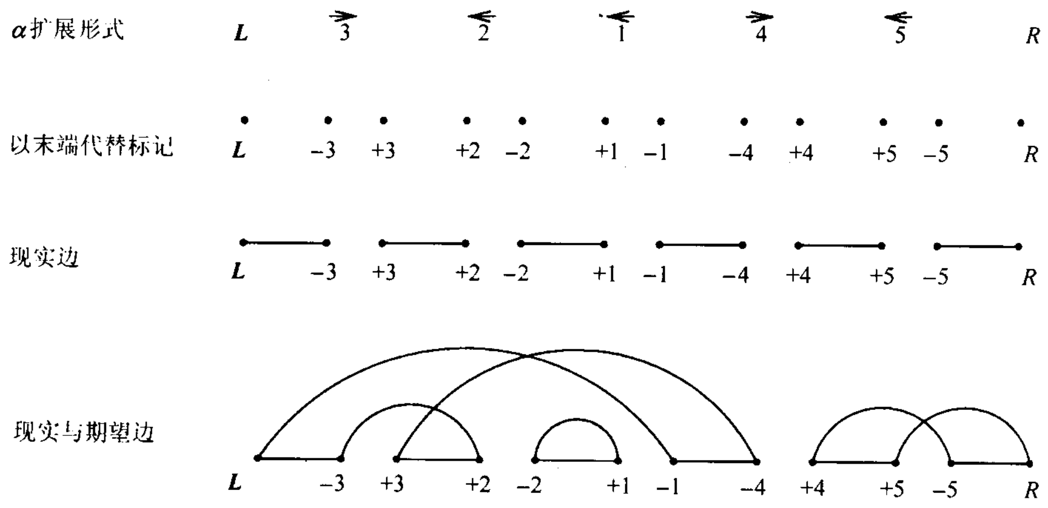


图 7.9　现实与期望图式的构造

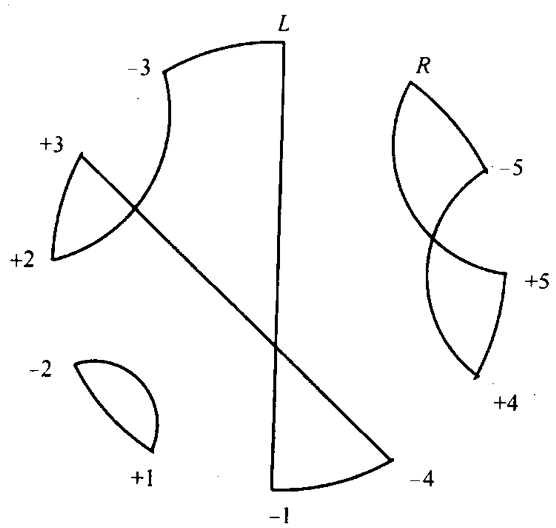


图 7.10　现实与期望图式

现实与期望图式有时称为现实与期望图(graph of reality and desire)，我们倾向于使用图式(diagram)一词是为了强调它需以特定的方式构造这一事实，而图可以以任何方式构造，只要我们正确地标记顶点并连接边线。但图式 $RD(\alpha)$ 必须按 α 排列中标记的次序绘制，这对于后面部分将要谈到的交叉期望边尤其重要。尽管这样，在谈到图式时我们仍旧使用图的标准术语，它们的语义是明显的。

对我们而言，一个重要特性是在 $RD(\alpha)$ 中每个结点的度都为 2，这是因为现实边集合 A 和期望边集合 B 在 $RD(\alpha)$ 中匹配，这意味着每一个结点恰好对应于 A 中的一条边和 B 中的一条边。并集 $A \cup B$ 由许多连通成分构成，每个成分是一个环，由 A 中的边和 B 中的边交替构成。而且，每个环有偶数个边，一半是现实边，另一半是期望边。图 $RD(\alpha)$ 中环的数量记为 $c_\beta(\alpha)$，或简记为 $c(\alpha)$(如果 β 是默认的)。接着注意到，因为 β
225 无断点，$c_\beta(\beta) = n + 1$，因此所有的环都由相同结点对之间的两条并行边构成。因为一共有 $2n + 2$ 个结点，所以有 $n + 1$ 个环，而且这也是满足 $c_\beta(\alpha) = n + 1$ 的惟一排列。

所以，在某种程度上，我们可把排序排列的过程看做一个把 $RD(\alpha)$ 转化为带尽可能

多环的图的过程。这并不必然意味着每一个增加环的数量的反转都有益，见习题 6。不过，在这里一个非常自然的问题是，一个反转如何影响 $RD(\alpha)$中的环？这正是下面的定理所要告诉我们的。首先注意到的一个特征是，反转在两个点之间切断当前的排列，该两点对应一条真实边。

定理 7.1　令(s,t)和(u,v)是排列 α 中界定一个反转 ρ 的两条真实边，(s,t)在(u,v)前面，则 $RD(\alpha\rho)$与 $RD(\alpha)$有如下不同：

1. 真实边(s,t)和(u,v)被(s,u)和(t,v)所替代；
2. 期望边保持不变；
3. 环中从结点 t 到 u 的反时针方向部分(包括 t、u 本身)被反转。

证明：对现实边的转化与 7.2.2 讨论的一样，惟一不同之处是我们这里处理的是末端而不是标记。在任何情况下这种处理都不影响到其他现实边。期望边没有发生改变，因它们仅依赖于目标排列。最后，末端的反转是反转作用于标记的直接结果。图 7.11 和图 7.12 显示了反转前和反转后的图。 226

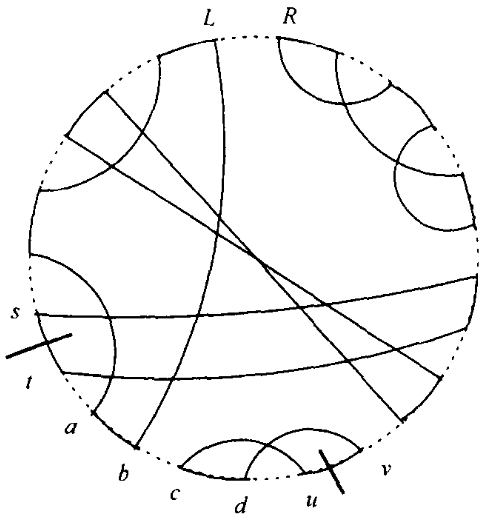


图 7.11　反转操作前的现实-期望图式，反转以两条现实边表示

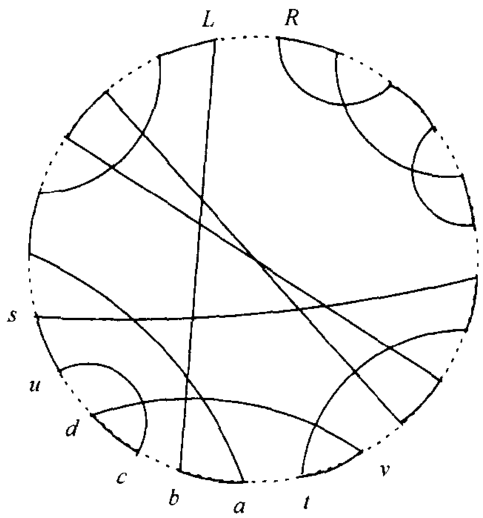


图 7.12　由图 7.11 获得的反转之后的现实-期望图

下面的定理告诉我们反转如何影响环的数量。在开始证明之前我们需要另外一个重要的定义。令 e 和 f 是 $RD(\alpha)$中属于同一环的两条现实边。暂时假定 e 和 f 在图中都是反时针的，其中任一条边均可导出环的朝向。如果 e 和 f 导出的朝向一致，就说它们是收敛的，反之则说它们是发散的。

每当我们需要把现实边考虑为有向边的时候，就把它们写成有序对，第一个成分是边的尾，第二个是边的头。我们下面将看到现实边的朝向对确定好的环是重要的。

定理 7.2　令 ρ 是作用于 $RD(\alpha)$上两个现实边 e 和 f 的反转，则：

1. 若 e 和 f 属于不同的环，则 $c(\alpha\rho)=c(\alpha)-1$；
2. 若 e 和 f 属于同一环并收敛，则 $c(\alpha\rho)=c(\alpha)$；
3. 若 e 和 f 属于同一环但发散，则 $c(\alpha\rho)=c(\alpha)+1$。

证明:图 7.13 将有助于澄清我们的论点。$RD(\alpha\rho)$中新的边以虚线箭头表示,边 e 和 f 的朝向如上所述。注意,我们没有把环画做图式的一部分,因为这里我们所需要的仅是 e 和 f 的朝向。

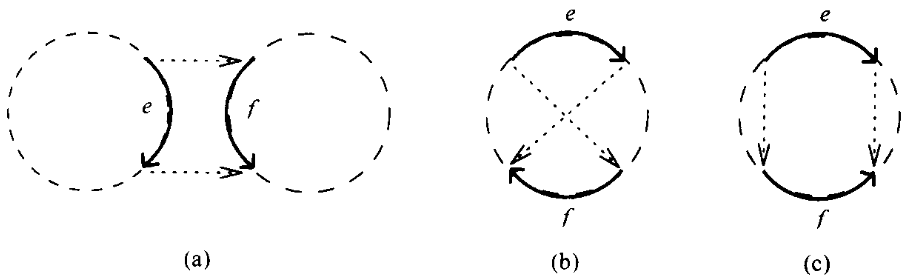


图 7.13　一个现实—期望图的环中反转的效应。(a) 来自不同环的边;(b) 收敛边;(c) 发散边

如先前所述,就涉及的环的数量而言,只有包含 e 和 f 的环才受影响,其他环作为连通组分不受影响。接着,我们分别讨论每种情况。

227 如果 e 和 f 属于不同的环,这两个环将融合成单个环[图 7.13(a)],所以环数减少 1。如果 e 和 f 收敛,新的边使环保持连通[图 7.13(b)],所以环数保持不变。若 e 和 f 发散,则环分为两片,导致环数增加 1[图 7.13(c)]。 ■

定理 7.2 有一重要结论,即每个反转至多将环的个数改变 1,这引出反转距离的另一个下限。

假定 $\rho_1, \rho_2, \ldots, \rho_t$ 为一系列使 α 转化为 β 的反转,则

$$\alpha\rho_1\rho_2\ldots\rho_t = \beta$$

通过计算环数,可得到

228
$$c(\alpha\rho_1\rho_2\ldots\rho_t) = c(\beta) = n + 1 \tag{7.5}$$

另一方面,由定理 7.2 有

$$c(\alpha\rho_1) - c(\alpha) \leqslant 1$$
$$c(\alpha\rho_1\rho_2) - c(\alpha\rho_1) \leqslant 1$$
$$\vdots$$
$$c(\alpha\rho_1\ldots\rho_t) - c(\alpha\rho_1\ldots\rho_{t-1}) \leqslant 1$$

相加所有项并抵消后得

$$n + 1 - c(\alpha) \leqslant t$$

若 $\rho_1, \rho_2, \ldots, \rho_t$ 是一个优化排序,则 $t = d(\alpha)$,且

$$n + 1 - c(\alpha) \leqslant d(\alpha) \tag{7.6}$$

这个下界非常好,它并不像 7.2.2 中所得到的那个。对大多数有向排列,这个下界与实际距离非常接近。它并不总是可行的原因将在下一部分解释。

7.2.4　交叉图

根据定理 7.2,我们可把 $RD(\alpha)$中的环分成好的与坏的。如果一个环有两个收敛的现实边,则它是好的,否则就是坏的。好的环可在现实与期望图式中被识别出,因为它们

至少有两条交叉的期望边。但我们必须小心，因为并不是所有有交叉期望边的环都是好环，交叉边仅表明环可能是好的。如果有疑问，则必须直接用定义检查，即我们应查找发散的现实边。交叉边特征强烈依赖于图式中环的排列。如果我们能以任何方式画图，则我们可随意让边交叉或不交叉。而在我们的图式中，期望边必须是弦（尽管有时因太接近圆周而被画得弯曲了一些），它们相对于其他弦的交叉特性是保守的。这种好与坏的分类不适用于仅有一条现实边的环，这种环不表示断点，在排序一个排列时也无需处理。我们把有至少 4 条边的环称为真环(proper cycle)。

如果在排列 α 中只有好的环，则下界(7.6)实际上是个等式，我们可排序 α，并按每次反转增加一个环的速度提高环的数量。当然，我们必须小心选择反转，后面将讨论这个问题。

即便存在坏环，即环中所有的现实边均导出相同的朝向，有时也可按每个反转一个环的速度进行排序。之所以能够这样是因为反转能够把一个坏环扭转(twist)成一个好环，裂解一个环的过程也正是构造另一个环的过程。因此每一步总是有一对发散的环，直至到达目标排列。不过，仅当某个环的期望边与另一个环的期望边交叉时我们才可能在裂解环的同时构造环。我们说两个环在这种情况下是交叉的(interleave)。注意，上述操作再次依赖于环中的期望边是弦这一事实。 229

验证 $RD(\alpha)$中哪些环与哪些环交叉非常重要。我们构造一个新的图，称做 α 关于 β 的交叉图(interleaving graph)，它以 $RD(\alpha)$中的真环作为结点，当且仅当对应的环交叉时两个结点邻接，这个交叉图记为 $I_\beta(\alpha)$。除了图的结构，我们还以好或坏标记每一个环。根据所含环的特性，$I_\beta(\alpha)$的连通组分被分成如下两类：如果一个组分全由坏环组成，则称之为坏组分；否则，称之为好组分。这样，具有至少一个好环的组分称为好组分。

下面给出好环、坏环和好组分、坏组分的例子。图 7.14 中的图式显示了一个 6 个环的排列，每个环有两条现实边和两条期望边。环 C 和 F 是好的环，其余为坏的环。好环

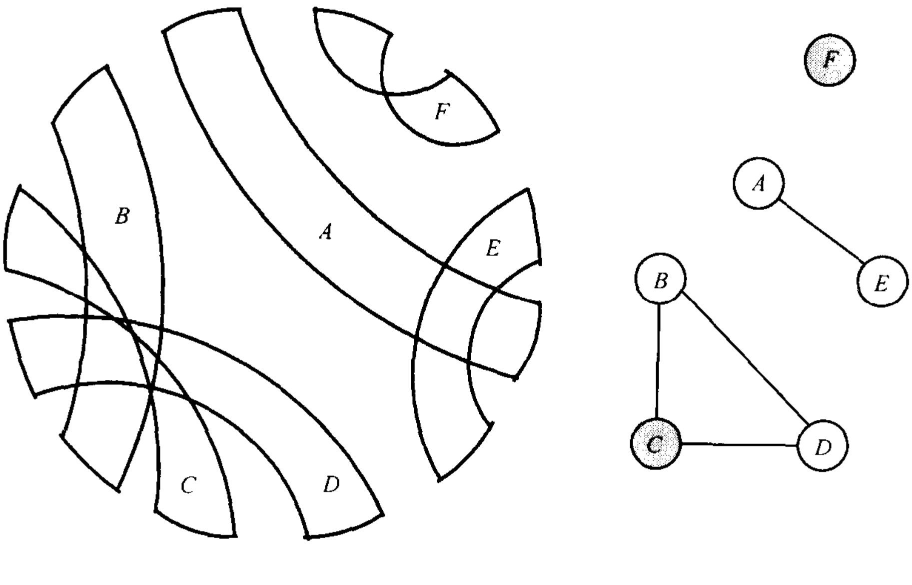
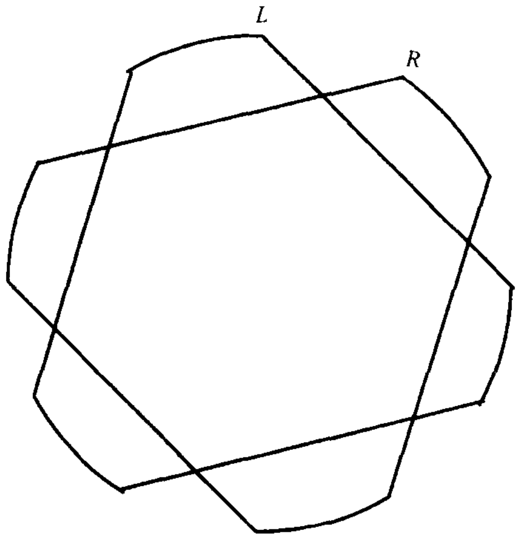


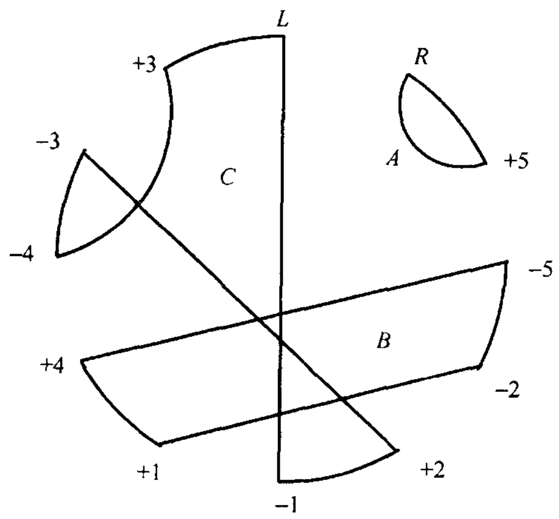
图 7.14 一个排列的现实与期望图式。环以字母 A 到 F 标记。这个排列的交叉图在右侧，好环以灰色表示

以灰色表示。该图还显示了这个排列的交叉图。在这个例子中有两个好组分和一个坏组分，这个惟一的坏组分是含坏环 A 和 E 的组分。

图 7.15 显示了含两个交叉坏环的图，图中没有好环，这意味着这个例子中不存在能增加环的数量的反转，且反转距离肯定大于下界 $n+1-c(\alpha)$。实际上这个论断可适用于任何含一个坏组分的图，因为作用于一个给定环 A 上的反转只能扭转与 A 交叉的环，所以只要反转作用于其他组分的环，这个坏组分就将一直是坏的。排序坏组分的惟一办法是使用一个不增加环的数量的反转。所以，对有坏组分的排列而言，下界 $n+1-c(\alpha)$
230 不是紧致的。

图 7.16 显示另一个例子，其中有两个真环 B 和 C，分别是坏环和好环。





231 图 7.15　一个不带好环的图式

图 7.16　一个有 3 个环的图式，注意只有真环构成交叉图

现在用最后一个例子来说明好组分的排序。回顾前面，我们需要选择同一个环中的两条发散边来定义一个增加环的数量的反转。在图 7.16 中，惟一的好环有三条现实边：$e=(L,+3)$，$f=(-3,-4)$和 $g=(-1,+2)$。因边 f 和 g 收敛，所以这一对不是一个好的选择。因边 e 和 g 发散，由它们定义的反转产生两个好组分，每个包含一个单环。同样 e 和 f 也发散，产生一个带两个环的好组分，这两种情况都没有坏组分的产生，因此，本例中有两个排序反转。

我们用下面的定理结束本节，它概括了迄今为止的结论，其证明仅是我们在例子中所用到论点的简单重复。

定理 7.3　由同一环的两条发散边所界定的反转是一个排序反转，当且仅当该反转不产生坏组分。

这条定理刻画了一个特定类型的反转，但并不告诉我们有关其存在性的信息。事实上，只要存在一个好环，就总是存在一个能增加环的数量的排序反转，但其证明从略。

7.2.5　坏组分

用 7.2.4 的方法排序了所有好组分后，我们现在转到坏组分的处理。本节的目的是将坏组分分成不同层次，并在此基础上提供一个计算给定排列关于目标排列的反转距离的确切公式。

坏组分按图 7.17 进行分类。首先区别**栅格**(hurdle)与**非栅格**(nonhurdle)；而对栅格则进一步区分**单栅格**(single hurdle)和**超栅格**(super hurdle)。这些概念在下面定义。

如果 $RD(\alpha)$中所有连通组分 A 中末端到组分 C 中末端的弦均与组分 B 中的期望边相交，我们就说组分 B 分隔 A 和 C。期望边也可看做弦，所以弦交叉的定义在此处亦适用。图 7.18 展示了这个概念，字母 A 到 F 表示组分，所有这些组分均为坏组分。在这个例子中，B 分隔 A 和 C。分隔的其他例子包括：E 分隔 F 和 C，B 分隔 F 和 D。习题 10 要求寻找该图中的所有分隔。

分隔是一个重要概念，因为作用于组分 A 与 C 中现实边的反转将导致分隔 A 与 C 的 B 组分被扭转。一个坏组分在扭转后可变为好组分，而一个好组分扭转后或保持为好组分或转变为坏组分。为避免问题复杂化，仅当没有好组分存在时我们才求助于扭转操作。

栅格是不分隔任何其他两个坏组分的坏组分。如果一个坏组分分隔其他坏组分，则它是一个非栅格。我们用 $h(\alpha)$表示 α 中栅格的数量。

我们说栅格 A 保护非栅格 B，如果移去栅格 A 将导致 B 变为栅格。换言之，每当 B 分隔两个坏组分时 B 是受 A 保护的。在图 7.18 中，F 保护 E，这是该图中的惟一的一例 232
保护。

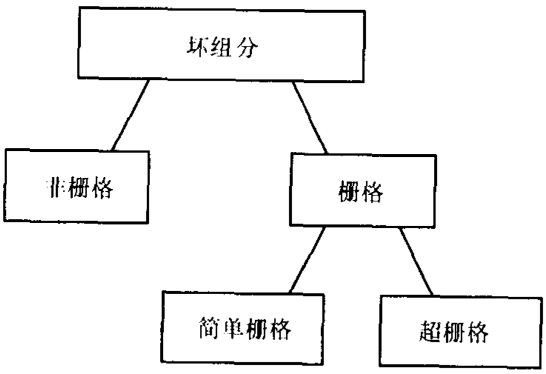


图 7.17　坏组分的分类

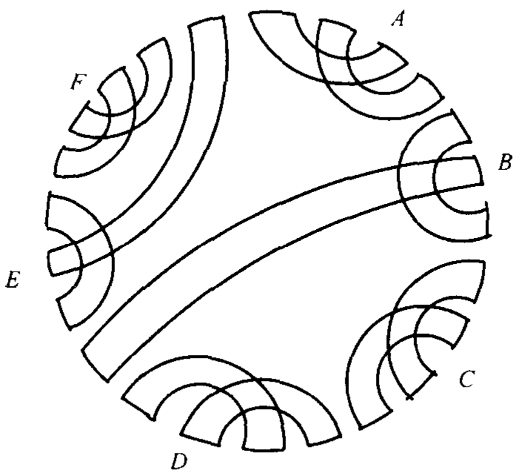


图 7.18　不含好环的一个图式，坏组分分隔其他坏组分 233

如果栅格 A 保护某个非栅格 B，则称栅格 A 为超栅格，否则称其为简单栅格。注意一个栅格最多仅能保护一个非栅格。在图 7.18 中惟一的超栅格是 F，其他栅格均为简单栅格。

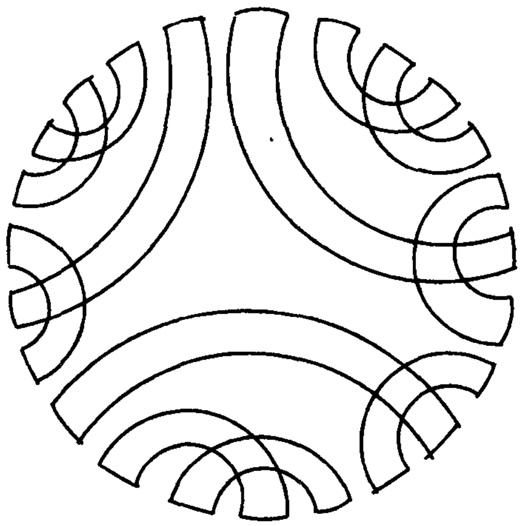


图 7.19　一个最小可能堡垒

当一个排列 α 的现实与期望图式含奇数个栅格且所有这些栅格均为超栅格时，称排列 α 为堡垒(fortress)。堡垒是一类排列，因为其特殊的结构，仅需一次反转即可排序。一个堡垒至少有 24 个标记，这是因为至少需要 2 个标记组成一个真环，4 个标记组成一个坏组分，8 个标记组成一个超栅格及其对应的受保护的非栅格，以及最后 3 个这样的超栅格及其伴随的栅格对组成一个堡垒。图 7.19 显示了这样一个最小可能堡垒的图。

我们现在可以写出有向排列反转距离的确切公式：

$$d(\alpha) = n + 1 - c(\alpha) + h(\alpha) + f(\alpha) \tag{7.7}$$

其中 $c(\alpha)$是环的数量(包括真环和非真环)，$h(\alpha)$是栅格的数量，$f(\alpha)$是 1 或 0(根据 α 是否是堡垒)。

项 $n+1-c(\alpha)$指好组分以及任何开始时为坏组分但在排序过程中变为好组分的组分。因坏组分有时需要一个额外的反转才能变成好组分，项 $h(\alpha)$是需要的。非栅格无需包括在公式中，因为它们将被扭转。最后，还需要一个额外的反转用于堡垒，因为在排
234 序过程中的某一刻非栅格会变为栅格。

7.2.6　算法

本节我们使用前面介绍的反转距离公式描述一个排序任意有向排列的算法，这个算法在每一步选择一个排序反转，因此能保证使用最少反转数量达到目标排列。

我们稍后将详细分析该算法的复杂性，现在先验证其正确性。在解释算法之前，先定义排序过程中用到的特殊类型的反转。

如同前面所见，当不存在好环时，我们被迫或者使用两个收敛边上的反转，或者使用不同环的边的反转。在前一情况下环的数量保持不变，而在后一情况环的数量减少一个。

当选择作用于收敛边上的反转时，作用于栅格上的边比作用于非栅格上的边要好，原因是这个操作把坏组分转变为好组分而不改变环的数量。如果能避开非栅格，则栅格的数量和堡垒的状态就不受影响，因此，根据式(7.7)，反转距离也保持不变。反转作用于超栅格上也不行，因为这样的话超栅格保护的非栅格会变成栅格，使 $h(\alpha)$保持不变。因此，每当我们必须对收敛边施行一个反转时，我们将把它作用在简单栅格上，而具体是简单栅格上的哪个环并不重要，所有的环在这一点上都是等价的。我们把作用于一个栅格中同一个环的现实边上的反转叫做**栅格切割**(hurdle cutting)。

当栅格是简单栅格时，栅格切割不改变 $c(\alpha)$，但减小 $h(\alpha)$。但我们必须小心，不然 $f(\alpha)$可能会增加。为消除这种 $f(\alpha)$增加的可能性，仅当 $h(\alpha)$是奇数时我们才使用栅格切割，这样，剩余的栅格便是偶数且我们确信得到的排列不是堡垒。因此，当栅格是简单

栅格且 $h(\alpha)$是奇数时，栅格切割是一种排序反转。

另一种可能性是对不同环的边使用反转，这会减少环的数量，即在反转距离中令 $n+1-c(\alpha)$增加。但如果能通过这个反转使 $h(\alpha)$减 2，我们仍能取得进展。做到这一点的惟一办法是选择不同栅格中的边，由不同栅格中的现实边界定的反转称为*栅格融合*（hurdle merging），其效果是这两个栅格以及分隔它们的任何非栅格将变为好组分。

栅格融合存在着将某个非栅格转变为栅格的危险性（见图 7.20）。当被融合的两个栅格像超栅格保护一个非栅格那样保护某个非栅格时，就会发生这种情况。为解决这个问题，我们需要一个额外的定义。当我们从 A 到 B 反时针方向遍历环所发现的栅格数与从 B 到 A 反时针方向遍历环所发现的栅格数相同时，就称两个栅格 A 和 B 是*对立的*（opposite）。注意，仅当 $h(\alpha)$是偶数时对立栅格才存在。选择对立栅格 A 和 B 进行融合是安全的，因为这保证了融合 A 和 B 时不会产生新的栅格，因此 $n+1-c(\alpha)+h(\alpha)$将减 1。

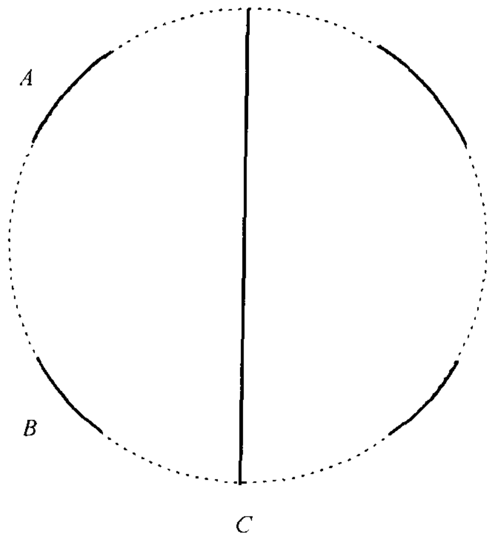


图 7.20　融合栅格 A 和 B 将使非栅格 C 转化为一个栅格。一个排序反转必须把 A 或 B 与某个栅格融合，A 或 B 与该栅格被 C 所分离

235

对 $f(\alpha)$也要小心。如果 $h(\alpha)$是偶数，则无需担心，因为与对立栅格融合的栅格将恰使 $h(\alpha)$减 2，保持其为偶数；如果 $h(\alpha)$是奇数且存在一个简单栅格，则不存在对立栅格，而我们使用一个排序栅格切割来得到一个偶数的 $h(\alpha)$；如果 $h(\alpha)$是奇数且不存在简单栅格，我们有一个堡垒，因而 $f(\alpha)$总是等于 1 且不会增加。可以得出结论，只要我们选用两个对立栅格做融合，栅格融合就是一个排序反转。

我们在图 7.21 的排序反转算法中总结了所有情况。对一个有向排列 α，函数 Sorting Reversal 找出一个排序反转。完整的算法仅由一个循环组成，在循环中我们调用该函数并施用所返回的反转，直到到达目标排列。

现在大体估计一下大小为 n 的排列所需的运行时间，我们从分析所有相关数据结构的构造开始。构建图式 $RD_\beta(\alpha)$需要线性时间，因为所需做的是确定现实与期望边。在这个图式中发现环需要 $O(n)$时间。对每个环我们需确定它是好的或坏的，每个环需要 $O(n)$时间，总共需要 $O(n^2)$时间。接着需确定环之间的交叉关系，做法是检查每一对期望边且把交叉图存储为邻接矩阵的形式，需时 $O(n^2)$。一旦获得关于好的与坏的环的足够信息，则可在线性时间里完成对好组分、坏组分、非栅格、简单栅格和超栅格的数量的计数（见习题 11）。因此，构造所有的数据结构需要的时间是 $O(n^2)$。

在排序反转算法中，最耗时的部分花在处理好组分上，我们需要尝试几次反转以判断所产生的排列的结构是否具有某种性质。由于一个反转由一对边确定，我们必须最多测试 $O(n^2)$次反转，对每一个都需耗费 $O(n^2)$的时间检查反转所产生的排列，所以在最坏的情况下排序反转的运行时间是 $O(n^4)$。为了排序一个关于 β 的排列 α，需要调用该函 236

数 $d_\beta(\alpha)$次，因此最终的时间是 $O(n^5)$。已有人提出更快的实现方法（见文献提要）。

```
Algorithm Sorting Reversal
    input: distinct permutations α and β
    output: a sorting reversal for α with target β
    if there is a good component in RD_β(α) then
        pick two divergent edges e, f in this component,
            making sure the corresponding reversal does not
            create any bad components
        return the reversal characterized by e and f
    else
        if h(α) is even then
            return merging of two opposite hurdles
        else
            if h(α) is odd and there is a simple hurdle
                return a reversal cutting this hurdle
            else
            // fortress
                return merging of any two hurdles
```

图 7.21　返回一个排序反转的算法，初始序列是 α，目标序列是 β，$\alpha\neq\beta$

7.3　无向块

基因组重排问题的另一种情况是基因组中块的方向是未知的，而仅知块排列的次序。对这种情况也有类似的理论，部分结果与有向块相同，但亦有许多结果与有向块不同。一个非常重要的差别是无向块排序导致 NP-难题，而有向块排序则可在多项式时间内求解。

我们由若干定义开始，大部分定义与有向块中给出的类似。一个无向排列 α 是一个从$\{1,2,\dots,n\}$到 n 个标记的集合$\mathcal{L}$上的映射，通常$\mathcal{L}$就是 α 的域。反转的定义与有向块相同，但不涉及箭头的翻转，因为无向块没有箭头，反转操作只是反转连续标记段的次序。
237 注意，不存在单个标记的反转，因为这样并不改变 α。

我们的目标是获取一个把 α 转化为 β 的最短可能反转序列。如果 $\rho_1,\rho_2,\dots,\rho_t$ 是一个最短反转序列，满足：

$$\alpha\rho_1,\rho_2,\dots,\rho_t = \beta$$

则 t 是 α 关于 β 的反转距离，记为 $d_\beta(\alpha)$。排序反转亦相应定义，有向块中的结论 $d_\beta(\alpha)=d_\alpha(\beta)$同样适用。如果根据上下文目标序列是明确的，我们便不再专门指明。本文中的恒等排列定义是对所有 $1\cdots n$ 中的 i 有 $I(i)=i$。下面的例子将澄清我们的讨论。

例 7.1　图7.22显示了两条带同源段的染色体，同源段夹在块之间，其中下面染色体的块以 1 到 8 标记，并转成上面染色体的块，同源块有同一标记。如此，我们得到了上层染色体的起始排列，而我们的目标是将它排序成下层的目标排列。一个能排序该排列的 4 步反转见图 7.23。

存在更短的反转序列吗？在本例中答案是否定的。为了证实 4 步反转是必要的，让
238 我们查看这个排列的断点，就像在有向排列中所做的。为简化起见，在本章剩余部分均假

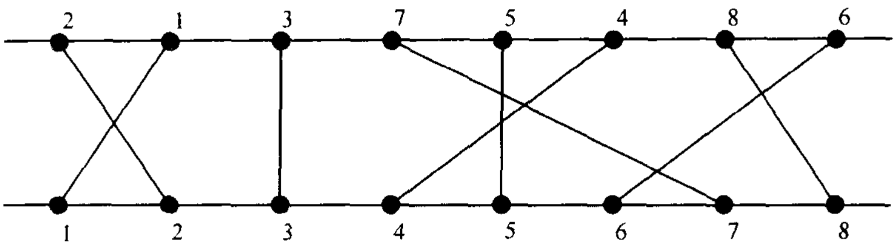


图 7.22　两个具有同源块的染色体

定目标排列是恒等排列。

无向排列 α 的一个断点是一对在 α 中相邻但在目标排列中并不邻接的标记,或等价地说,它们是一对相邻而不连续的标记。我们在 α 的扩展形式中考虑断点,包括额外的左边元 L 和右边元 R。$b(\alpha)$表示 α 中断点的个数。例子中的排列有 7 个断点,如图 7.24 所示。非断点的标记对是 2,1 和 5,4。

2　1　3　7　5　4　8　6

1　2　3　7　5　4　8　6

1　2　3　4　5　7　8　6

1　2　3　4　5　7　6　8

1　2　3　4　5　6　7　8

图 7.23　排序排列 21375486

正如有向排列一样,一次反转最多可移去两个断点。因此,我们先前由有向排列得到的下界公式这里仍然适用,即:

L • 2　1 • 3 • 7 • 5　4 • 8 • 6 • R

图 7.24　以黑点"•"表示的无向排列的断点

$$d(\alpha) \geqslant \frac{b(\alpha)}{2} \tag{7.8}$$

根据这个不等式,很容易看出为什么我们不能够以少于 4 次的反转排序图 7.24 中的 α 排列。根据式(7.8)有 $d(\alpha) \geqslant 3.5$,而因为 $d(\alpha)$是一个整数,所以实际上 $d(\alpha) \geqslant 4$。因此我们的 4 次反转的解已是最优的。

7.3.1　条

考虑如下的排列 α 的扩展形式,断点以黑点表示:

$$L \bullet 4 \quad 5 \bullet 3 \quad 2 \quad 1 \bullet R$$

在两个连续断点之间有一系列连续的标记,或升序或降序。略加思考我们知道这实际上是一个一般现象。确实,如果有两个不构成断点的相邻标记,它们必须是这种形式:

$$\cdots x(x+1)\cdots$$

或

$$\cdots x(x-1)\cdots$$

在第一种情况下,如果 $x+1$ 不是断点的一部分,则其右侧的标记必是 $x+2$,这是因为另一个连续标记 x 已用过了,因此上升的标记序列向右侧扩展,直至达到下一个断点或排列的末尾。类似的论断可用于第一种情况 x 的左侧,或上面第二种情况的 x 的两侧。 239

被断点包含而无内部断点的连续标记序列称为条(strip)。有两类条:递增条和递减条。一个被断点包含的单个标记本身也是条,它既是递增的又是递减的条。标记 R 和 L 是这个规则的例外,它们总是被考虑为递增条的一部分,尽管它们总是自成一条。事实上,为了定义条,我们将这些端点元素看做单个条。例如,如果 0 属于递增条 $0,1\cdots$,而 $n+1$ 是更长的条 $\cdots,n,n+1$ 的一部分,我们把这两个序列考虑为一个单个条,它们由公共元素 $L=R$ 连接。

例 7.2　下面是5个条的例子,2 个递增条($L=R,1,2$ 和 $5,6$),1 个递减条($8,7$),2 个既是递增又是递减的条(3 和 4)。

$$L\quad 1\quad 2\bullet 8\quad 7\bullet 3\bullet 5\quad 6\bullet 4\bullet R$$

定理 7.4　如果标记 k 属于一个递减条,而 $k-1$ 属于一个递增条,则存在一个至少可移走一个断点的反转。

证明:标记 $k-1$ 和 k 不能属于同一条,因为只有单个的标记才可既是递增条又是递减条。因此它们属于不同的条。反过来这意味着每个标记是其所在条的最后一个元素,即每个元素都立即紧跟了一个断点,如图 7.25 所示。

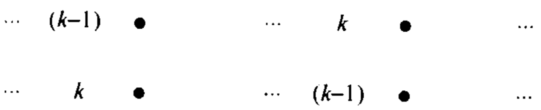


图 7.25　k 和 $k-1$ 的条的可能相对位置

这个图呈示了两种可能的机制。在排列中,$k-1$ 的条要么在 k 的条的前面,要么在其后面。在两种情况下,消除紧跟 $k-1$ 和 k 的两个断点的反转把两个标记接合在一起,因此可至少把断点数量减 1。 ■

我们知道,所有的排列都至少有 1 个递增
240 条,它包含 L(或 R)。相比而言,一个排列可以不包含递减条。不过,如果一个排列有一个递减条,则依据前面的结果存在一个可移去断点的反转。之所以会如此是因为,当既存在递增的又存在递减的条时,由于它们在恒等排列中的次序,它们必须在某处相遇。另一种处理的方式是,令最小的标记属于一个递减条,称其为 k,则 $k-1$ 必属于一个递增条,若不然 k 将不是最小的。若在以上陈述中用 $k+1$ 替代 $k-1$,类似的结果仍然成立,如下面的定理所示。

定理 7.5　如果标记 k 属于递减条而 $k+1$ 属于递增条,则存在一个至少能移去一个断点的反转。

证明:如前一个定理一样,k 和 $k+1$ 不能在同一个条中,因此它们各自为某个条的第一个元素,前面是一个断点。剪切这两个断点的反转可减少断点的总数,因为 k 和 $k+1$ 将一并结束。 ■

这两个结果意味着,只要有递减条我们总是能减少断点数。注意这也适于单元素的条。不过,有时我们没有递减条。当我们必须创建一个排列而在排序过程中无递减条时,下面的结果提供了一个补偿方法。

定理 7.6　令 α 是一个含一个递减条的排列。如果所有从 α 中移去断点的反转均不留下递减条,则存在一个移去 α 中两个断点的反转。

证明:令 k 是递减条中的最小标记。如在定理 7.4 的证明中所见,有一个反转 ρ 可连接 k 和 $k-1$。如果 $k-1$ 在 k 的右侧,则 ρ 留下一个递减条,这与我们的假设矛盾,因此 $k-1$

必然在 k 的左侧[见图 7.26(a)]。

(a) … $(k-1)$ • … k • …

(b) … • l … • $(l+1)$ …

图 7.26 不带留下递减条的反转的排列

现在令 l 为一个递减条的最大标记。由定理 7.5,存在一个联合 l 和 $l+1$ 的反转 σ。如果 $l+1$ 在 l 的左侧,则 σ 留下一个递减条,因此 $l+1$ 必然在 l 的右侧[见图 7.26(b)]。注意到 k 必须在由 σ 反转的区间内,否则 σ 对 k 的递减条没有作用。类似地,l 必 241
须在 ρ 的区间内。

我们断言这两个区间实际上是一样的,即 $\rho=\sigma$,如果不是这样的话,则存在一个条仅属于某一个反转的区间。假定该条属于 ρ 的区间,若该条是递增的,则 ρ 使之成为递减条;若它是递减的,则它在 $\alpha\sigma$ 中仍旧保存下来,因为 σ 只能增大它。因此这两种情况都不可能成立。如果该条属于 σ,类似的论断亦可得出,所以断言是成立的。但在本例中这个反转可移去两个断点,因为它融合 $k-1$ 和 k 以及 l 和 $l+1$。 ■

7.3.2 算法

我们现在可给出排序一个排列的算法,使用的反转数至多为最少反转数 $d(\alpha)$ 的两倍,因而这是一个近似算法。它依赖于前面的定理,其蕴含的想法非常简单。存在一个主循环,用于按某些条件查找当前排列和选择最可能的施用于该排列的反转。然后更新当前排列并报出所使用的反转。当前排列为恒等排列时循环停止。

我们用于选择反转的标准基于前面的结果。如果当前排列有一个递减条,则寻找一个能减少断点数量并留下一个递减条的反转。较好的候选反转是在定理 7.4、7.5、7.6 的证明中出现的反转。若无这类反转,则我们知道存在一类包含所有递减条并移去两个断点的反转。最后,如果当前排列没有递减条,则选择能切割两个断点的反转。我们知道至少存在两个断点,否则当前排列是恒等排列。不存在仅有一个断点的排列,所以所有与恒等排列不同的排列至少有两个断点。更进一步,一个消除两个断点的反转保证不会增加断点的总数。算法的结果见图 7.27。

我们知道这个算法会停止,因为断点数量随每次迭代而减少,除非遇到无递减条的排列,而这种情况在同一行中不会发生两次。因此我们可以肯定地说,经每两次迭代断点的数量至少减 1。事实上,如同下面的定理所揭示的,存在一个更大的断点平均减少率。

定理 7.7 在排序无向排列算法中(图 7.27),迭代的次数小于或等于初始排列中断点的个数。

证明:我们的目标是证明每一步迭代平均至少移走一个断点。当不存在递减条时,惟一的困难来自对反转的选择。但每当我们遇到这种情况时,前面的迭代恰巧移走了两个断点,维持了每一迭代平均至少移走一个断点的论断。如果我们恰好遇到初始排列是不带递减条的,则这一论断对初次迭代就不适用。但在最后一个迭代,即产生恒等排列的迭代中,该迭代移走两个断点,补偿了这种特殊情况。 ■ 242

Algorithm *Sorting Unoriented Permutation*
  input: permutation α
  output: series of reversals that sort α
  list $\leftarrow$ empty
  while $\alpha \neq I$ **do**
    if α has a decreasing strip **then**
      $k \leftarrow$ the smallest label in a decreasing strip
      $\rho \leftarrow$ the reversal that cuts after k and after $k-1$
      if $\alpha\rho$ has no decreasing strip **then**
        $l \leftarrow$ the largest label in a decreasing strip
        $\rho \leftarrow$ the reversal that cuts before l and before $l+1$
    else
      $\rho \leftarrow$ the reversal that cuts the first two breakpoints
    $\alpha \leftarrow \alpha\rho$
    list $\leftarrow$ *list* $+ \rho$
  return *list*

图 7.27　一个排序无向排列的算法,它能保证所用反转的数量不大于最小反转数的两倍

小　　结

生物的遗传物质随着进化而改变。相对于序列水平的点突变,本章中我们关心的是基因组水平上的较大的变化,主要研究一种常见的转化,即反转。给定两个基因组,我们寻找能够将一个基因组转化为另一个基因组的最小反转数,这就是它们之间的反转距离。反转距离是两个基因组之间差别的一种最节俭的解释。

根据可获得的基因朝向信息,这一问题有两个版本:一个是处理有向排列的,另一个则是处理无向排列的。一个值得注意的事实是,尽管前者的反转距离可在多项式时间内计算,后者却是一个 NP-难题。

我们介绍了一个漂亮的理论,它引出有向排列的多项式时间算法。该理论依赖于简单的概念,诸如断点、现实与期望图,以及更复杂的概念如栅格和堡垒。

对于无向排列,我们发展了一系列工具,提供有关实际距离的重要信息,然后介绍了
243 一个能够产生一系列反转的近似算法,反转数量至多是优化反转的反转数量的两倍。

习　　题

1. 考虑下面的 DNA 片段:
   5′…CCTCACTAGCGACCAGACTATAAATTACGTGACCTGT…3′
   5′…GGAGTGATCGCTGGTCTGATATTTAATGCACTGGACA…3′
   其中“基因”a = GGTCA,b = AATTAC,c = AGTCTGG,d = CACTAG。找出这两个基因在 DNA 片段中的出处,并写出它们在$\{a,b,c,d\}$上的一个排列,描述基因的位置和方向。
2. 找出 $\overleftarrow{a}\overrightarrow{b}\overrightarrow{c}\overrightarrow{d}\overleftarrow{e}$相对于 $\overrightarrow{a}\overrightarrow{b}\overrightarrow{c}\overrightarrow{d}\overrightarrow{e}$的断点。

3. 有两个排列 α = ●□□●□●□□，β = ●□●□□●□□。有人擦掉了标记，仅留下 α 相对于 β 和 β 相对于 α 的断点。你能在空位上写出使断点保持如上的有向标记吗？
4. 已知 $RD_{\beta}(\alpha)$有下列边，重构排列 α 和 β：
   现实边：$(L,-3)$，$(-1,-2)$，$(R,+1)$，$(+2,+3)$。
   期望边：$(L,-1)$，$(-2,+1)$，$(R,+3)$，$(-3,+2)$。
5. 考虑有向排列，显示一个能移走两个断点却不是排序反转的反转。
6. 考虑有向排列，显示一个能增加 $RD(\alpha)$中环的数量却不是排序反转的反转。
7. 证明下面的论断：一个有向排列容许一个反转，该反转增加环的数量当且仅当它至少有一个边带有由右到左的箭头。假设目标排列是恒等排列。
8. 证明在现实图与期望图式中，每一条不与现实边重合的期望边必然与某些期望边相交。
9. 证明对任何有向排列 α 和目标排列 β，$b_{\beta}(\alpha)/2 \leqslant n+1-c_{\beta}(\alpha)$。
10. 找出坏组分的所有的三元组 XYZ，使得在图 7.18 中 Y 分离 X 和 Z。
11. 假定已知好、坏环和交叉图的信息，说明如何在 $O(n)$时间计算非栅格、简单栅格和超栅格的数量。
12. 已知有向排列和它的目标排列，你是否能找出它们所有的排序反转？
13. 证明任何含 n 个元素的无向排列可用最多 $n-1$ 次反转完成排序。
14. 反转排序所有在$\{1,2,3,4\}$上的 24 个无向排列，其中有多少个需要 3 次反转？
15. 找出一个无向排列，它不能够仅靠可切开断点的反转来被优化排序。
16. 无向排列 α 的在$[1\cdots n]$上的一个节是区间$[i\cdots j]$，使得对每一个 $k\in[i\cdots j]$有 $i\leqslant j$ 和 $i\leqslant\alpha(k)\leqslant j$。注意$[1\cdots n]$对每一个排列都是一个节。证明，如果$[1\cdots i]$对某个 $i<n$ 是 α 的一个节，则 $d(\alpha)\leqslant n-2$。
17. 在$[1\cdots n]$上的一个 Golan 排列是排列 α，其惟一的节是$[1\cdots n]$，且对所有的 $i\in[1\cdots n]$有$|\alpha(i)-i|\leqslant 2$。 244
    a. 说明 Golan 排列的逆是一个 Golan 排列。
    b. 说明对每个 $n\geqslant 3$ 在$[1\cdots n]$上仅有两个 Golan 排列，其中一个是另一个的逆。
    c. 说明如果 α 是一个 Golan 排列且 $n\geqslant 4$，则 $b(\alpha)=n+1$。

**18. 说明在$[1\cdots n]$的 Golan 排列 α 满足 $d(\alpha)=n-1$。

**19. 说明 Golan 排列是在$[1\cdots n]$的反转距离为 $n-1$ 的惟一排列。

文献提要

本章的大部分内容是依据 Bafna 和 Pevzner[18]、Hannenhalli 和 Pevzner[87]，以及 Kececioglu 和 Sankoff[116]的工作编写的。你会注意到我们的注释与这些文章有很大的不同，特别是我们用箭头替代了加号和减号，且我们的处理中没有衬垫(padding)。另外，在涉及排列时，我们用术语“有向(ordered)”和“无向(unordered)”分别代替这些文献中的“无符号(unsigned)”和“有符号(signed)”。

Berman 和 Hannenhalli 对图 7.21 的排序反转算法提出了更有效的实现[23]。Han-

nenhalli 和 Pevzner 对涉及多个染色体集合的基因组重排问题提出了一个多项式时间的解[88]。Ferreti、Nadeau 和 Sankoff 等研究了根据哪个基因出现在哪个染色体上的有关信息来推断进化的问题[62]。

Sankoff 给出了另一种基因组进化的分析方法[167]。Waterson、Ewens 和 Hall 等研究了排序循环排列的问题,并提出了许多猜想[205]。

直接或间接讨论基因组重排的文献包括:Koonin 和 Dolja 对 RNA 病毒的研究[117],Palmer 的文章[148],Palmer、Osorio 和 Thompson 等对叶绿体 DNA 的研究[150],Fauron 和 Havlik 对玉米线粒体基因组的比较[58],Palmer 和 Herbon 对芸苔家族线粒体基因组的比较,其中包括芜菁和卷心菜[149],以及 Hoffmann、Boore 和 Brown 对软体动物线粒体 DNA 的研究等[95]。

Caprara、Lancia 和 Ng 等提出了一种排序无向排列的算法[31],这个算法在实际应用中有良好的效果。Caprara 最近表明这个问题是 NP-难题。Gates 和 Papadimitriou 研究了一个相关的问题,其中排序必须应用前缀反转来完成[69]。

第 8 章　分子结构预测

迄今为止我们对生物分子的研究仅限于其基本线性序列。但从第一章即可看出，像 RNA 或蛋白质这样的一个生物分子实际上具有三维结构。由于这个结构决定了生物分子的功能，理解生物分子的空间结构便成为分子生物学的最重要的目标。使用 X 射线晶体衍射和核磁共振确定生物分子的三维结构技术难度高、耗资大且并非总是可行，这与确定生物分子的一级结构（如 RNA 中的碱基和蛋白中的氨基酸）的相对容易性形成了鲜明对照，其后果是形成了目前的序列知识与结构知识之间的鸿沟。我们已知道几千种蛋白质的序列，但仅知道几百种蛋白质的三维结构，且该鸿沟每年都在不断扩大。这种情况催生了结构预测方法的研究，这些方法能够根据分子的一级结构（序列）预测其三维结构，使用包括从分子动力学模拟到神经网络程序的各种方法。当然，在本书中，我们首先感兴趣的是算法技术，该类技术尚未达到一般性地预测分子结构的地步，因为算法依赖于分子折叠的精确模型，而这种精确模型迄今尚不存在。但毫无疑问，可以给出在某些特定情况下的算法公式，其应用价值已得到多年实践的证实，例如下一节将讲述的 RNA 结构预测的例子。本章还将讨论蛋白质的分子折叠问题，该问题的求解是分子生物学的“圣杯”。我们描述了该问题，并给出了一种很有前景的算法途径。

8.1　RNA 二级结构预测 246

回顾一下第一章的内容，RNA 是由 A、C、G、U 四种核苷酸组成的单链。RNA 的单链特性导致它与 DNA 不同，分子中的一部分核苷酸可与其他部分核苷酸互补配对，即 RNA 分子可产生折叠。而且，核苷酸序列惟一决定了分子折叠的方式，这也是我们通过分析核苷酸序列预测分子结构的依据。对于当前的预测方法，真正的三维结构还是过于复杂，所以我们主要专注于二级结构预测，实际上是试图确定哪一些碱基相互配对。

我们可把 RNA 分子看做是 n 个字符的串，$R = r_1 r_2 \cdots r_n$，$r \in \{A,C,G,U\}$，典型地，n 为几百至几千。分子的二级结构是碱基对 (r_i, r_j) 的集合 S，其中，$1 \leqslant i \leqslant j \leqslant n$。原则上要求 r_i 与 r_j 互补，且 $j - i > t$，其中 t 为阈值，因为 RNA 不能以过于尖锐的角度折叠。而预测方法是基于对最小自由能构象的计算，所以可通过对对应构象施加较大自由能来

避免上述的尖锐折叠的情况。

需排除一种叫做结(knot)的自然构象。当$(r_i, r_j) \in S, (r_k, r_l) \in S$,且 $i<k<j<l$ 时,存在结。没有结 S 就变为平面图。结在 RNA 分子中确实存在,而排除结可简化问题。排除结的另一个理由是,二级结构可用于预测三维结构,而结可在后一阶段再予以考虑。

根据上述基础,我们不禁要问:什么样的算法适用于预测 RNA 的二级结构? 在回答该问题之前,应先更严谨地考虑这个问题。前已述及,我们要计算最小自由能结构,故此需找出某种方法将能量值赋予一给定的结构。我们简单假定已存在一个或几个经验函数可完成这一任务,因此最简单的算法便是:枚举所有可能的结构,选择有最低自由能的那个。不幸的是,可能的结构数量与碱基数呈指数关系,故就大的分子而言这种算法并不实用(值得注意的是,部分研究者已建立并使用了这种枚举的方法)。所以我们需要寻找其他更有效的方法确定最小自由能结构。下面介绍两个公式,它们能产生实现这一目标的简单动态程序设计算法。

= = = = = = = = = = = 独立碱基对 = = = = = = = = = = =

对用于寻找最小自由能结构的第一个公式,须假定在所有的 RNA 结构中碱基对的能量是相互独立的。这虽是一种粗略的近似,但有用且有助于我们理解更完善的算法。
247 在这个假定条件下,结构 S 的总自由能可定为:

$$E(S) = \sum_{(r_i, r_j) \in S} \alpha(r_i, r_j) \tag{8.1}$$

式中 $\alpha(r_i, r_j)$为碱基对(r_i, r_j)的自由能。我们约定,当 $i \neq j$ 时,$\alpha(r_i, r_j)<0$;若 $i=j$,$\alpha(r_i, r_j)=0$。

因为可用小序列串的解来确定大序列串的解,碱基对能量独立的假定成为提供有效算法的关键。例如,假定要确定串 $R_{i,j}=r_i r_{i+1} \cdots r_j$ 中的二级结构 $S_{i,j}$的最小自由能。考虑发生在碱基 r_j 上的情况。i 与 j 之间的某一碱基要么形成配对,要么不形成。若不配对,则 $E(S_{i,j})=E(S_{i,j-1})$;若 r_j 与 r_i 配对,则 $E(S_{i,j})=\alpha(r_i, r_j)+E(S_{i+1,j-1})$。最后,如果 r_j 与某个 r_k 配对,$i<k<j$,此时我们可把串分拆成两个:$R_{i,k-1}$和 $R_{k,j}$,并说对某个 k 有$E(S_{i,j})=E(S_{i,k-1})+E(S_{k,j})$(这样做的前提是假定没有结形成)。问题是如何知道 k 取值为何时总能量最低? 因此,问题变成 $E(S_{i,j})=\min\{E(S_{i,k-1})+E(S_{k,j})\}$,$k$ 在i 和j 之间。这样做的意义在于可用较短的串的结构来计算 $E(S_{i,j})$。这种情形我们是熟悉的,它意味着我们可用动态程序设计技术来求解该问题。特别是,上述等式可以写做下面的形式,这与第三章中的递归关系非常相似:

$$E(S_{i,j}) = \min\begin{cases} E(S_{i+1,j-1}) + \alpha(r_i, r_j) \\ \min\{E(S_{i,k-1}) + E(S_{k,j})\} \quad for i < k \leqslant j \end{cases} \tag{8.2}$$

[j 无碱基配对的情况被上述第二式包括,因为当 $k=j$ 时有$E(S_{j,j})=0$]。获得 $E(S_{1,n})$的算法与在 3.2 节中看到的算法非常类似(填充 $n \times n$ 矩阵的值),这里不再重复。但需要注意的是,与 3.2 节一样,递归求解结果为我们提供了一个最低自由能值,而实际的结构需通过对动态程序设计矩阵的某些后处理才能得到,正如我们通过序列比较矩阵获得

比对一样。

让我们估算上述算法的运行时间。从 3.2 节的基本序列比对算法可以看出矩阵的每一单元花费了常数时间，因此 $n \times n$ 矩阵运行时间为 $O(n^2)$。此处情况略有不同，如式(8.2)所示，对每对 i, j 必须使用数量不同的先前矩阵入口进行求和，确切的求和数为：

$$\sum_{j=1}^{n} \sum_{i=1}^{j-1} (j - i - 1) \tag{8.3}$$ 248

该算法的运行时间为 $O(n^3)$。

=========== 有 环 结 构 ===========

现在放宽对碱基对自由能彼此独立的假设，认为碱基对的自由能依赖于相邻碱基对，这导致需考虑 RNA 的环(loop)结构。假定 S 中有碱基对 (r_i, r_j)，考虑位置 u、v、w，$i < u < v < w < j$。若对任何 u 和 w 碱基对 (r_u, r_w) 都不是 S 中的碱基对，则碱基 r_v 是从 (r_i, r_j) 可达的(accessible)。环是从碱基对 (r_i, r_j) 可达的所有碱基的集合，图 8.1 显示了各种不同类型的环。没有结的假定意味着任何结构 S 可把串 R 分解成不相交的环。

环的大小是其所含的不配对碱基的个数。我们把环的概念拓展到环的大小为零的情况。当几个碱基对互相邻接，便产生所谓的"堆积碱基对"或"螺旋区域"。因此一个结构中环的个数由其所含的碱基对个数给出。

一些结构的出现与否对自由能具有影响。一个例子是螺旋区域具有负的(稳定的)自由能，而大于零的环有正的(不稳定的)自由能。结果是，当 (r_i, r_j) 是闭锁一个环的碱基对时，结构 $S_{i+1,j-1}$ 可能不是子串 $R_{i+1,j-1}$ 的最小自由能结构。换言之，可能存在结构 $S'_{i+1,j-1}$，其自由能低于 $S_{i+1,j-1}$。但当我们把碱基对 (r_i, r_j) 的能量加到 $S_{i+1,j-1}$ 时，稳定的自由能可能会反超。这意味着除了式(8.2)，我们尚需给出新的递归关系式。

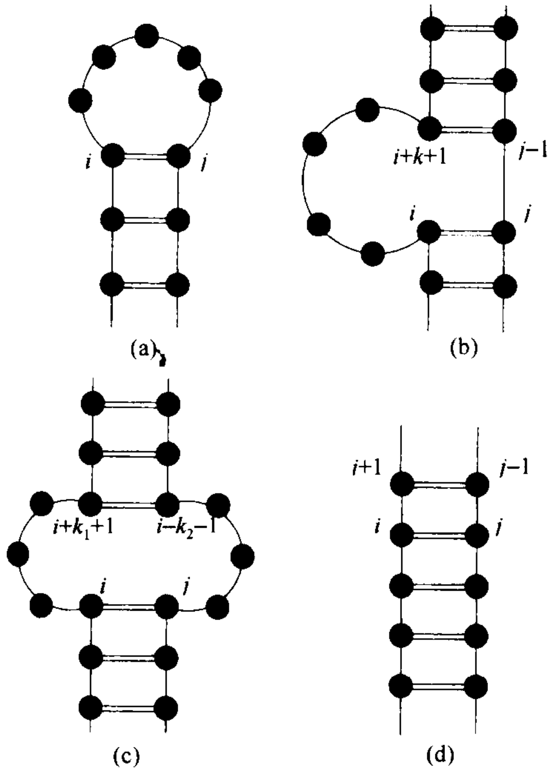


图 8.1　无结的 RNA 二级结构。黑点表示核糖核苷酸，水平双线表示碱基对。(a)发夹环；(b) i 位的凸；(c)内环；(d)螺旋区域(取自文献[105])

相对于完全一般的环模型，这些递归关系仍包含了若干的简化，这些简化对产生更有效的动态程序设计算法是必要的。一个一般的模型可导致一个指数运行时间的算法。

第一个简化是不考虑不属任何环的碱基的自由能；第二个简化是结构的总自由能是各组分自由能简单累加之和；最后一个简化是假定仅考虑图 8.1 所列出的各类环。

我们现在确定串 $R_{i,j}$ 的结构 $S_{i,j}$。首先假定 r_i 不是配对的碱基，则 $E(S_{i,j}) = E(S_{i+1,j})$。假定 r_j 也不配对，类似地，$E(S_{i,j}) = E(S_{i,j-1})$。然后假定 r_i 和 r_j 是配对的

碱基,但不彼此配对,则 $E(S_{i,j})=\min\{E(S_{i,k})+E(S_{k+1,j})\}$,$i<k<j$,$r_k$ 是 i 和 j 碱基之间的任何碱基。最后一种情况是(r_i,r_j)形成配对,这意味着 i 和 j 之间可有一个或多个环,令 $L_{i,j}$表示在此情况下的最小自由能结构。汇集全部的递归关系,得到:

$$E(S_{i,j}) = \min\begin{cases} E(S_{i+1,j}) \\ E(S_{i,j-1}) \\ \min\{E(S_{i,k}) + E(S_{k+1,j}), i < k < j \\ E(L_{i,j}) \end{cases} \tag{8.4}$$

现在讨论 $L_{i,j}$的详细情况。我们必须考虑图 8.1 的各种类型的环,因此需表达各种自由能的值。如前所述,我们假定实验已确定了这些自由能的值并已表示为函数的形式,
249 且 $\alpha(r_i,r_j)$给出了碱基对(r_i,r_j)的自由能。另外,还有下列这些自由能函数:

$\zeta(k)$ = 大小为 k 的发夹环的不稳定自由能

η = 相邻碱基对的稳定自由能

$\beta(k)$ = 大小为 k 的凸的不稳定自由能

$\gamma(k)$ = 大小为 k 内环的不稳定自由能

用上面的函数,可得 $E(L_{i,j})$的递归关系式:

$$\alpha(r_i,r_j) + \xi(j-i-1) \quad \text{若 } L_{i,j} \text{ 是发夹环} \tag{8.5}$$

$$\alpha(r_i,r_j) + \eta + E(S_{i+1+,j-1}) \quad \text{若 } L_{i,j} \text{ 是螺旋结构} \tag{8.6}$$

$$\min_{k\geqslant 1}\{\alpha(r_i,r_j) + \beta(k) + E(S_{i+k+1,j-1})\} \quad \text{若 } L_{i,j} \text{ 是 } i \text{ 位的凸} \tag{8.7}$$

$$\min_{k\geqslant 1}\{\alpha(r_i,r_j) + \beta(k) + E(S_{i+1,j-k-1})\} \quad \text{若 } L_{i,j} \text{ 是 } j \text{ 位的凸} \tag{8.8}$$

$$\min_{k_1,k_2\geqslant 1}\{\alpha(r_i,r_j) + \gamma(k_1+k_2) + E(S_{i+k_1+1,j-k_2-1})\} \quad \text{若 } L_{i,j} \text{ 是内环} \tag{8.9}$$

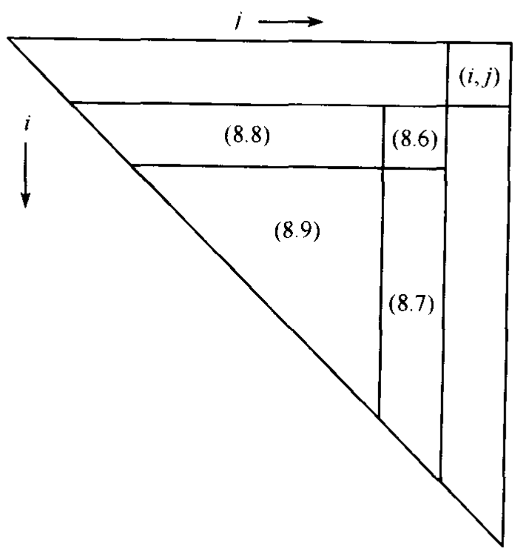


图 8.2 递归关系式 8.6 到 8.9 计算方法的矩阵表示。等式数字标明了需知道先前的 $E(L_{i,j})$值的矩阵区域

$E(L_{i,j})$值可取上述递归关系式计算结果的最小值,这些公式,再加上式(8.4),可给出建立预测有环 RNA 二级结构的基于矩阵的动态程序设计算法所必需的信息。图 8.2 所示的矩阵结构说明了在计算 $E(L_{i,j})$时哪些区域的矩阵单元需事先已知。

因算法最复杂的部分是 $E(L_{i,j})$递归关系的计算,故我们先进一步研究这些关系式。最简单的情况是 $L_{i,j}$是螺旋或发夹结构,在这种情况下,每对(i,j)所耗费的时间是常数,因此运行时间为 $O(n^2)$。在凸出现的情况下,参数 k 介于 1 和 $j-i$ 之间,因此类似于无环情况下最小自由能的计算,运行时间为 $O(n^3)$。最后一种情况是内环,这需处理两个参数 k_1 和 k_2,这种情况下必须查看 E 的先前值,它们出现于由 i 和 j 限定的子矩阵中。直觉上,这意味着对每

一对 i、j 我们必须先查找有 $O(n^2)$ 个元素的子矩阵,因此总的运行时间为 $O(n^4)$。更形式地,对于某些常数 c,我们有

$$\sum_{1\leqslant i\leqslant j\leqslant n}\left(\sum_{i\leqslant i'\leqslant j'\leqslant j} c\right)=O(n^4),$$

对特定的 i、j、k_1 和 k_2,该常数的值由 $\alpha(r_i+r_j)+\gamma(k_1+k_2)+E(S_{i+1+k_1,j-1-k_2})$ 的计算时间确定。 251

= = = = = = = = = = = 提高内环计算效率 = = = = = = = = = = =

从前面的框架可清楚地看出,计算内环耗费时间最多。我们现在说明这个运行时间如何可被降低到 $O(n^3)$。沿着对角线一个环的大小是常数,利用这一事实我们可做到降低运行时间。下面是一个例子。

上述 γ 函数的参数是内环的大小,即 k_1+k_2。对 k_1 和 k_2 的约束条件是 $2\leqslant k_1+k_2\leqslant j-i-2-l_{\min}$,这里 $l_{\min}$ 是最小环的大小。假定要计算 $E(L_{4,18})$,且 $l_{\min}=1$,则 $2\leqslant k_1+k_2\leqslant 11$。接着,在某一点我们能有 $k_1=2$ 和 $k_2=6$,意味着我们需要计算 $\gamma(8)$ 和 $E(S_{10,14})$,以及在另一点有 $k_1=5$ 和 $k_2=3$,意味着我们需要计算 $\gamma(8)$ 和 $E(S_{7,11})$。这样,(10,14)和(7,11)位于同一对角线上,且环的大小是常数。

该例揭示了下面的技巧:对每一对 (i,j),我们存储一个向量 $V_{i,j}$,其下标为环的所有可能的大小。存在至多 $2n-1$ 个对角线,因此这个向量最多含同样数量的元素。在每一 252
元素中存放针对每一对角线的最小值。所以,当对一个内环以及给定的碱基对 (i,j) 计算 $E(L_{i,j})$ 时,仅需对环的大小 l 的所有可能值计算 $\{\alpha(r_i,r_j)+\gamma(l)+V_{i,j}[l]\}$ 的最小值。因此,每对 (i,j) 需花费 $O(n)$ 的时间,总运行时间为 $O(n^3)$。为了能运用该方法,我们需有一个辅助矩阵来存放所有的向量 $V_{i,j}$,且在扫描原始矩阵 M 时必须更新这些向量。所幸的是,可渐进式地做到这一点。例如,当从行 i 移到 $i+1$ 时,行 $i+1$ 中的每一个元素对应 $V_{i,j}$ 中的一个元素(M 中的一条对角线)。通过将 M 的 $i+1$ 行的元素与 $V_{i,j}$ 中的元素相比较,可得到 $V_{i+1,j}$ 的元素,运行时间为 $O(n)$。因此,我们亦耗费 $O(n^3)$ 的时间更新 $V_{i,j}$ 向量。所有这些意味着我们可消除内环计算运行时间中的一个因子 n,但储存空间需是原来的 2 倍。

如果仔细审视所用的各种自由能函数,运行时间还可进一步改善,类似于第三章中的空隙罚分函数计算时间的改善。鉴于形成的算法相当复杂,此处不详述,读者可参考文献提要。需指出的是,对于线性的不稳定函数,上述的运行时间可进一步优化到 $O(n^2)$(对于内环和凸)。实验证明不稳定函数是呈对数增长的。

上述的 RNA 结构预测算法有一重大缺陷,即它们仅能预测出一个结构。在实际中,我们希望有一组的解,它们中的某些可能在自由能上是次优的。这样做的原因同本书中讨论的其他问题类似:给定用于建立算法的模型的抽象,最优的解(或任意优化解)未必是真正的结构。出于这一原因,在某个能量阈值下,扩展上述算法以便发现优化的以及各种次优化的解是很重要的。对于这种扩展,本章后的文献提要列出了一篇参考文献。

8.2 蛋白质折叠问题

蛋白质折叠问题的主要目的是根据蛋白质的氨基酸序列确定其三维结构。这个问题自20世纪50年代早期起就被广泛研究,但到目前尚无满意的解决方法。对这一问题已有大量的工作,本节仅作一简要的介绍,下一节则介绍一个特定的方法。我们相信这两节的材料已足以使读者对该问题有一个起步性的认识。进一步阅读的参考文献列在章尾。

对于所有预测蛋白质折叠的方法而言,一个重要的假定是氨基酸顺序完全且惟一地决定了折叠。该假设有下面的实验证据支持:如果在体外无任何其他物质存在的条件下
253 使蛋白质去折叠,然后复性,蛋白质将立刻重新折叠回原来的三维结构,整个折叠过程少于1秒。所以蛋白质获得三维结构的信息已完全包含在氨基酸序列中(但并非对所有的蛋白质都是这样,部分蛋白质需要辅助分子才能折叠)。

在讨论蛋白质折叠预测问题的困难性之前,先简要介绍蛋白质三维结构的主要内容。回想第一章,我们知道蛋白质有不同水平的三维结构,在线性的一级结构之上是二级结构,包括 α 螺旋、β 片层以及既非螺旋也非片层的环3种基本类型。

α 螺旋是一种简单的螺旋,每个螺旋平均含10个氨基酸残基,或3个螺旋转折。某些螺旋可有多达40个残基。研究表明某些氨基酸相比于其他氨基酸更多地出现在螺旋中,这有助于二维结构的预测,但并不足以得到准确的结果。

一个 β 片层由几个氨基酸序列片段相互结合组成,每一参与部分称为一个 β 链,一般有5到10个残基。这些链彼此邻接,形成一种扭转的片层。某些氨基酸可优先形成 β 链,但同样地这种优先性并不足以产生精确的预测。

环是连接在上述两种结构之间的氨基酸片段。不同于螺旋和片层,环在形状和大小上都不是规则的结构。通常环在折叠的蛋白质的外部,其他结构组成蛋白质的**核心**(core)。进化上相关的蛋白质通常具有相同的螺旋和片层,但环的结构可以很不同。在这种情况下,核心片段是保守的。核心的概念对下一节非常重要。

蛋白质还有更高级的结构,称做*基序*(motif)和*结构域*(domain)。基序是指出现在不同蛋白质中的几个二级结构的简单组合。一个基序的例子是螺旋-环-螺旋结构(helix-loop-helix),研究表明该结构是钙离子的结合位点,所以该基序有明确的功能。其他基序好像无特定功能。结构域是更复杂的结构,由几个二级结构组成,且必须有特定的功能,包含一个活性位点。活性位点是指能结合外部分子的蛋白质部分。一个蛋白质可有一个或几个结构域。上述内容组成了蛋白质的三级结构。

我们现在更详细地叙述折叠问题。给定一蛋白质的氨基酸序列,我们想知道的是哪些部位形成 α 螺旋、β 片层与环,以及其基序和结构域如何组织。假定对每一氨基酸已有足够和可靠的物理和化学信息,原则上就应能计算出折叠的自由能。一种预测方法是简单地枚举所有可能的折叠,计算每一种折叠的自由能,并选择具有最低自由能的那种(假定该种折叠是蛋白质的天然结构,事实上这并没有得到完全确认)。现在我们看一看一种典型的蛋白质具有多少种可能的折叠形式。回忆1.2节所讲述的蛋白质能够折叠主要归功于 α 碳原子与邻近的原子形成的键角 ϕ 和 Ψ。实验证明这些键角有彼此独立的几个数

值,每一个残基的每一对键角对应于一个构象。让我们假定每一个键角仅可取 3 个值,这 254
意味着每一个氨基酸残基有 9 种构象。对于一个小的有 100 个氨基酸残基的蛋白质,存在 9^{100} 个可能的构象,这个天文数字排除了枚举算法的可行性。

更糟糕的是,即便研究者同意有最低自由能的蛋白质其结构是天然结构,也不存在计算构象能量的统一方法。尚有许多的因素需要考虑,如形状、大小、不同组分分子的极性、分子水平的相互作用力等。按目前的知识,我们仅能得到一些一般的规则,如疏水性氨基酸在蛋白质的内侧和亲水性氨基酸在蛋白质的外侧。

这种状况导致研究者专注于蛋白质二级结构的预测方法,而上述将自由能与三级结构关联起来的困难性亦适用于二级结构。因此,尚无一般而可靠的动态程序设计办法可用于蛋白质二级结构预测。实际中所用的是各种不同方法的混合,从基于神经网络的程序到基于氨基酸残基统计性质的模式识别方法。然而,这些方法仍没有达到科学家所期望的可靠程度,因此人们仍在发展其他技术来解决这个问题。下一节要讨论的就是其中的一种技术。

8.3　蛋白质比配

一个用于蛋白质二级结构预测的早期方法基于这样的概念,即相似的序列应具有相似的结构。如果我们已知蛋白质 A 的结构(比如利用 X 射线晶体衍射),若蛋白质 B 的序列与之十分类似,则有理由推断 B 的结构应与 A 类似。不幸的是,通常这并不正确。在序列水平上相似的蛋白质可具有完全不同的二级结构。另一方面已观察到序列水平差别很大的蛋白质可在结构上相关联,尽管有不同的环,但有类似的核心。这些观察导致研究者提出逆向蛋白质折叠问题,不是通过序列来预测结构,而是试图用已知的结构适配序列。

为使结构适配序列,我们必须更精确地定义“结构”。在本节中,结构实际上是指结构模型(structural model),这种模型可通过下述过程得到。给定一结构已知的蛋白质,通过用替代物替换氨基酸,同时保持替代物与原始氨基酸在基本特性上的关联性,我们可得到结构模型。这些特性因所采用的模型而不同,但它们均保持原有的 α 螺旋、β 片层及环等
氨基酸的原有结构并反映这些结构的空间限制,如与其他氨基酸的距离、同位置的替换物 255
有多少在结构内部及有多少在结构外部等。我们这里不给出这种模型的细节,但假定局部结构的这种特征是存在的。

对于一个给定的结构模型,我们可以用未知结构的蛋白质与之进行比对,考虑那个特殊模型所禀赋的性质。这种特殊的比对称为*蛋白质序列结构比配*(protein threading,以下简称蛋白质比配)。我们将看到这个技术导致一个精确的组合优化难题,因为有许多种方法可将一个蛋白质序列与一个模型比配。正如本书中的其他例子一样,这也是一个 NP-难题,但这个问题可通过一种广为所知的解决 NP-难题的技术来解决,这一方法所取得的成功也是编写本节的动机。下面更详细地描述该问题。

我们假定有下列方式的输入:

1. 由 n 个氨基酸 a_i 组成的蛋白质序列 A;

2．核心结构模型 C，有 m 个核心片段 C_i，该模型含下列信息；

a．每一核心片段的长度为 c_i；

b．核心片段 C_i 和 C_{i+1} 通过环区域 λ_i 连接，已知每一 λ_i 的最大（$l_i^{\max}$）和最小（$l_i^{\min}$）长度；

c．此模型中每一位置的氨基酸的局部结构环境（化学性质、空间限制等）。

3．用于估算一个给定结构序列比配的计分函数。

期望的输出是一个整数集合 $T=\{t_1,t_2,\ldots,t_m\}$，其中 t_i 的值表示占据核心片段 i 中第一个位置的氨基酸是 A 中的哪一个氨基酸，这清楚地表明蛋白质比配是序列与核心结构模型之间的一种比对。

在这个公式中，计分函数显然起了重要的作用。给定了一个比配和局部结构环境，计分函数应能评估每个氨基酸在模型中的位置以及不同氨基酸之间的相互作用。这种相互作用可以是两两之间、三个或四个之间、或更多氨基酸之间的相互作用。

若忽略氨基酸之间的相互作用，并允许有长度可变的环区域，则该问题可通过标准的动态程序设计技术来解决。若允许成对的相互作用，问题则变为 NP-难题（见文献提要）。与其对问题进行过分简化使之回到熟悉的动态程序设计方法，我们在这里宁可允许成对相互作用，并展示如何用一个处理 NP-难题的标准技术来切实求解这个问题。这种技术也叫做**分支界限法**（branch-and-bound）。现在，我们暂离主题，先解释这种方法，看如何应用它来解决成对相互作用条件下的比配问题。

＝＝＝＝＝＝＝＝＝＝＝分枝和界限＝＝＝＝＝＝＝＝＝＝＝

在一个组合优化问题中，我们要在解空间的许多可能性中找出最佳解。所谓最佳指
256 的是对每个候选解计算某个函数 f 获得的最大或最小值。尽管对 P 类问题最优解可在多项式时间内找到，但对大多数问题解空间是呈指数增长的。在 NP-难题的情况下，不存在已知算法能在多项式时间内搜索解空间找出优化解。因此，惟一能肯定地获得优化解的办法是枚举并评估每一个可能解，然后择取最佳的。不过，应用下面的方法我们能稍稍加快这个过程。

首先，根据某些约束条件将解空间分成几个子空间是可能的。例如，给定解空间 S，我们有可能将它分解为两个子集 S_1 和 S_2，S_1 由所有具有某种性质的解组成，而 S_2 由所有不含这种性质的解组成，这就是分枝界限法中的分枝。

其次，对于一集合中的所有候选解，可得到优化函数 f 取值的一个界限范围，这个界限可用来筛除许多解。事情是这样来考虑的：假定有一最小化问题，且我们已知一些候选解 s 的值 $f(s)$。现在考虑解空间的子集 X，且假定我们可以获得 X 中任一解的下限值 l。最后，假定 $l>f(s)$，这意味着 X 中的任一解都不能比已获得的解更好，应被舍弃。而若 $l<f(s)$，则需研究集合 X，因为最佳的方案可能就存在在那里。

这样我们看出，在用分枝界限法进行最小化的算法中，关键部分是把解空间进行有效分割和找到所有候选解的最佳函数值的下限。此下限应尽可能地接近实际函数值，因为这样我们就能更快地找到最佳解。另外，它还需便于计算。然而需记住应用分枝界限法求解 NP-难题的最差运行时间总是指数的。

现在将这些概念应用于蛋白质比配问题。首先需要定义所有可能的解，为此应先列出每个可能的比配所面临的约束。空间的约束可由下面的不等式给出：

$$1 + \sum_{j<i}(c_j + l_j^{\min}) \leqslant t_i \leqslant n + 1 - \sum_{j \geqslant i}(c_j + l_j^{\min})$$

另一个约束来自这样一个事实，即任何比配须遵循模型中核心片段的次序（这个条件在更一般的情况下可以放宽），这导致：

$$t_i + c_i + l_i^{\min} \leqslant t_{i+1} \leqslant t_i + c_i + l_i^{\max}$$

这些约束条件提示每一个 t_i 必须位于 b_i 和 e_i 确定的区间之间，因此一组可能的比配可由一个区间集合表达，每个 t_i 一个区间（图 8.3）。

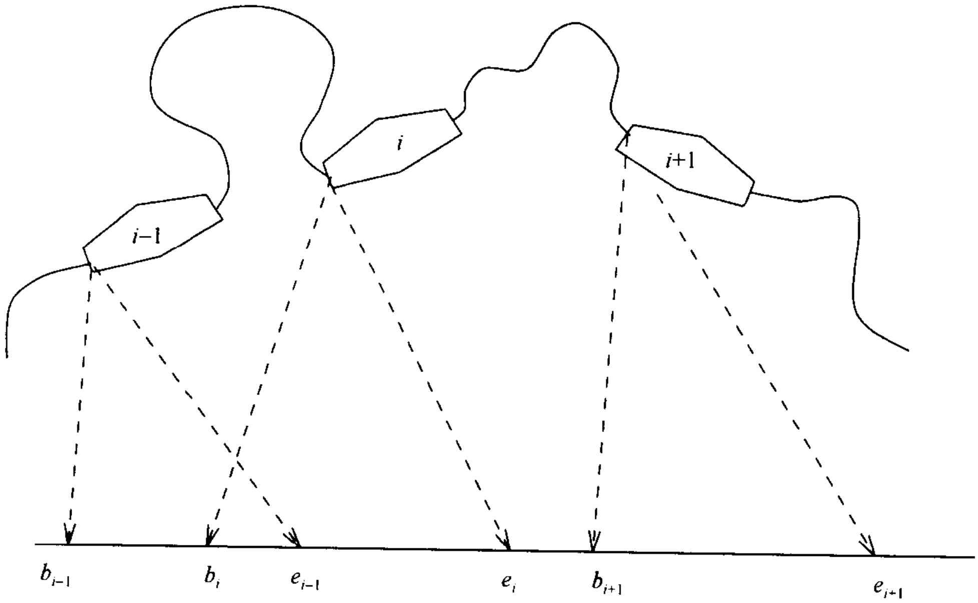


图 8.3　一组比配。核心片段以多边形表示，标为 $i-1$、i 和 $i+1$，在其间的是环。下面的线表示被结构匹配的序列，图中虚线箭头表示的是每个核心片段的第一个位置取值的区间，所以列出了几组可能的比配

现在解释算法的分支部分。选择一个核心片段，比如 i，以及区间 $[b_i, e_i]$ 中的位置 u_i。把解集分为三子集：在第一个子集中 t_i 的新区间是 $[b_i, u_i - 1]$，在第二个子集中 t_i 的新区间是 $[u_i + 1, e_i]$；最后一个子集中 t_i 的值被强制成 u_i。我们没有规定选择哪一个 257
核心段，也没有限制核心段在区间内的位置，这些选择可根据区间的大小来确定。

现在考虑下限。假定 T 是某个比配，计分函数可由下式得出：

$$f(T) = \sum_i g_1(i, t_i) + \sum_i \sum_{j>i} g_2(i, j, t_i, t_j) \tag{8.10}$$

注意，计分函数处理核心片段而不是单个氨基酸，原因是 t_i 的值决定了核心片段 i 中序列 A 的所有氨基酸的位置，因此自变量对 (i, t_i) 封装了所有有关片段 i 的信息。上式右边的第一项给出了每个核心片段对整体结构的贡献，而第二项给出每一核心片段对的贡献。函数 g_1 和 g_2 是输入。

给定一组比配 $\mathcal{T}$，一个简单下限可由下式给出：

$$\min_{T\in\mathcal{T}} f(T) = \min_{T\in\mathcal{T}} \sum_i \left[g_1(i, t_i) + \sum_{j>i} g_2(i, j, t_i, t_j) \right] \tag{8.11}$$

258
$$\geqslant \sum_i \left[\min_{b_i \leqslant x \leqslant e_i} g_1(i, x) + \sum_{j>i} \min_{\substack{b_i \leqslant y \leqslant e_i \\ b_j \leqslant z \leqslant e_j}} g_2(i, j, y, z) \right] \tag{8.12}$$

这个下限通过累加各项的下限之和得到。可能存在更强的下限(见习题 7)。最好获得一个对应于一特定比配的下限,因为这个比配接着可被用于解空间的分解过程。假定比配 T 当前有一个下限的最低值,我们从中选择一个核心段 C_i,并在 u_i 点分裂它,所选的核心段可能是具有最大区间值的那一个。算法的描述如图 8.4 所示。

```
Algorithm Protein Threading
  input: protein sequence, core structural model, score functions,
      lower bound function f_lb
  output: a threading T
  𝒯 ← all possible threadings
  lb ← f_lb(𝒯)
  // We use a priority queue Q to keep threadings
  // In Q each element is a set of threadings and its lower bound
  Insert(Q, 𝒯, lb)
  while TRUE do
      // We remove a candidate set of threadings from Q
      𝒯_c ← RemoveMin(Q)
      if |𝒯_c| = 1 then
          return 𝒯_c
          // The only remaining threading is the solution
      else
          Split(𝒯_c)
          for each new subset 𝒯_i from 𝒯_c do
              lb_i ← f_lb(𝒯_i)
              Insert(Q, 𝒯_i, lb_i)
```

图 8.4　分支界限法蛋白质比配算法

至此我们完成了用于蛋白质比配的分支界限法的描述。我们用有关该技术使用的若干评论来结束本节。首先,这是一个新近发展的领域,因此尚需许多测试。这些测试可大致分为三类:第一类是,当从一个结构获得了一个结构模型时,要进行自比配(self-threading),也就是对产生该结构模型的蛋白质本身进行比配;第二类测试使用遗传上相近的蛋白质(因而具有相似的序列),它们具有相似的结构(称做同源蛋白质,protein homolog);最后一类测试使用序列不同但结构相似的蛋白质(称做结构拟似体,structural analog)。在这些测试中所有的结构都是事先已知的,因此我们可准确地知道测试的结果并可测知异常。当获得的结构十分接近正确结论时,就说得到了好的测试结果;当存在错误时,可容易地施加额外的约束以校正它们,这也是该技术的另一优点。由此可利用经年累积的
259 有关蛋白质结构的各种知识。

该技术的实际应用情形应是这样:维持一个蛋白质结构模型数据库;研究者在得到一个新的蛋白质序列后将之提交到数据库服务器;服务器然后将此序列与数据库中的所有模型进行比配,返回一至几个适配的模型。注意,这意味着对数据库中每一个结构模型求

解一个蛋白质比配问题,这要求对每一个这样的问题有一个快速的解。

小　结

本章我们介绍了代表性的分子预测算法。我们研究了两种用于预测 RNA 二级结构的动态程序设计算法,并展示了如何将更复杂算法的运行时间由 $O(n^4)$改进为 $O(n^3)$。我们接着简要解释了蛋白质折叠问题,指出了求解该问题的许多途径,其中的一个方法称为蛋白质比配。基于处理 NP-难题的熟悉技术,该方法可有一个算法实现。

习　题

1. 给定 RNA 序列 AUGGCAUCCGUA,找出一个具有最大碱基对数的结构。
2. 给出一个出现结的 RNA 序列的例子。
3. 具体写出 RNA 序列预测的动态程序设计算法,其中每个碱基对的能量是彼此独立的。
4. 给出一个算法,使用式(8.2)的递归关系检索 RNA 二级结构。
5. 用于 RNA 二级结构预测的不稳定函数的种类会影响动态程序设计算法的运行时间。使用线性、对数、平方函数或简单查询表作为例子解释为什么会这样。
6. 证明蛋白质比配是 NP-完全问题。
7. 给出一个更强的联配计分函数下限。
8. 列出本书中的其他 NP-难题,并设计一个分枝界限算法。

*9. 计算式(8.10)的下限需要计算落入某区间内的所有点(g_1 的例子),或计算落入某区域内的所有点(g_2 的例子)。如何预先处理所有这些点以高效地回答这些查询? 260

文献提要

整个 8.1 节的组织基本上是基于 Zuker 的论文[207],该文包含了许多额外的资料。Waterman 和 Smith 给出了计算中内层循环计算时间的改进[201]。对于特殊函数的计算时间的进一步改进是由 Eppstein、Galil、Giancarlo[55]以及 Larmore、Schieber 作出的[121]。对确定的能量阈值计算优化与次优化结构的工作由 Zuker 完成[206]。关于 RNA 结构预测一般方面的一篇综述是 Turner 和 Sugimoto 的论文[186]。

已有大量的关于蛋白质折叠问题的文献,这里我们指出几个最近的重要的综述。Richards 的文章是针对一般读者的[161],Creighton 的综述侧重于技术方面[38]。蛋白质折叠中不同分子间交互所起的作用见 Dill 的论文[48]。我们对蛋白质折叠的讨论部分地基于 Kanehisa 和 DeLisi 的文章[105],其中的两个图取自该文。Unger、Moult 和 Fraenkel 研究了蛋白质折叠的计算困难性[189,65],他们证明基于简单网格模型(lattice model)获得最小能量蛋白质折叠是一个 NP-难题。Berger[21]以及 Berger 与 Wilson[22]给出了一个蛋白质基序识别应用的成功例子。

蛋白质比配部分基于 Lathrop 和 Smith 的工作[123]。图 8.3 来自他们的论文中的一幅类似的图。该文包括大量关于测试的材料,以及用于加速计算的各种技术的详细描述。Lathrop 证明带各种长度的环的蛋白质比配问题和成对氨基酸交互问题是 NP-完全性的[122]。

第 9 章　结语:DNA 计算

全书中列出了不少解决分子生物学的算法。本章介绍从相反方向出发所得到的令人惊奇的结果:如何利用分子生物学解决困难的计算问题。这些结果催生了一个令人兴奋的新领域——DNA 计算,其最终目标是创造高效的生物分子计算机。尽管这种计算离实用尚远,但大量的研究正在进行以克服其不足,因此不久的将来必会出现一个新的发展。

9.1　哈密顿通路问题

本节介绍如何应用 DNA 分子在有向图中求解哈密顿通路问题。在该方法中,实验 261
操作的时间与图的顶点的数量呈线性增长。我们在 2.3 节中介绍了哈密顿通路问题是 NP-完全问题,这里的方法并不是一个 P = NP 的证明,因为该方法本质上是一个执行指数数量操作的强制性算法。该方法能够在一个线性的实验操作时间里完成是因为指数数量的操作是并行执行的。

形式上,待求解的问题是这样的。给定有向图 $G=(V,E)$,$|V|=n$,$|E|=m$,以及两个不相邻的顶点 s 和 t,验证是否存在通路$(s,v_1,v_2,\ldots,t)$,其长度(边的数量)是$n-1$,且所有的顶点不同。我们把该问题称做 HDPP,即哈密顿有向路径问题(Hamiltonian directed path problem),图 9.1 是一个例子。强制算法十分简单,即产生恰好 $n-1$ 条边的所有通路,并验证是否存在一条符合问题的给定条件。至多存在$(n-2)!$ 条通路,因
为对通路的第一个顶点而言存在$(n-2)$个可选择顶点,而第二个顶点又有$(n-3)$个可选 262

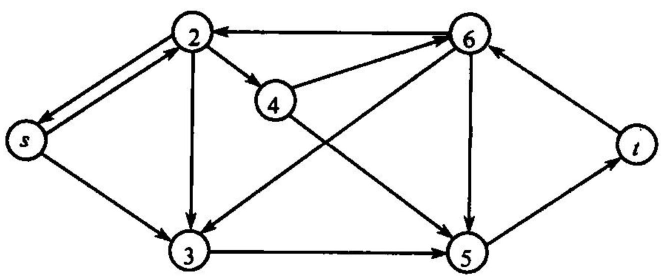


图 9.1　在一个有向图中,一个 st-哈密顿通路是$(s,2,4,6,3,5,t)$

择顶点,如此等等。

下面呈示的方法并不是按上述方法产生的,而是先用第一步产生数量巨大的随机通路,这一办法使得所有通路能够并发地产生。另一方面,这也使得算法概率化。完整的算法如下:

1. 产生随机通路;
2. 在所产生的通路中,仅留用那些始于 s 终于 t 的通路;
3. 在步骤 2 保留的通路中,保留恰好经过 n 个顶点的通路;
4. 在步骤 3 保留的通路中,保留那些对每个顶点至少访问一次的通路;
5. 如果此时尚存任何通路,答案就是 yes,反之则是 no。

现在我们描述上述步骤的每一步是如何用实验完成的。

=========== **随机通路的生成** ===========

假定我们可获得长 20 个核苷酸的随机单链 DNA 序列,而产生天文数字(超过 10^{10})的短链 DNA 在实际中很容易做到。$\bar{S}$ 代表 DNA 串 S 的逆补。

- **顶点表示**(vertex representation):随机选择 n 个不同的单链 20 字符的 DNA 序列,每一个顶点 v 与序列 S_v 关联;对每一序列获得其逆补 $\bar{S}_v$。在试管 T_1 中产生 $\bar{S}$ 的许多拷贝;
- **边表示**(edge representation):若$(u,v)\in E$,通过拼接 S_u 的 10-碱基后缀和 S_v 的 10-碱基前缀构建序列 S_{uv},后缀和前缀靠 DNA5′-3′方向相区别。当 $u=s$ 时 S_s 的整个都参与拼接,S_{sv}有 30 个碱基;当 $v=t$ 时,情况类似。注意在这种拼接下
263 $S_{uv}\neq S_{vu}$。接着,在试管 T_2 中生成 S_{uv}序列的许多拷贝;
- **通路构造**(path construction):将试管 T_1 和 T_2 倒入到 T_3,我们假定在 T_3 中序列间的结合可自由发生,也就是说 T_3 中将发生许多**连接酶**反应(ligase reaction)。

通过执行上述步骤我们可以获得许多随机通路。原因何在?我们通过例子进行解释。对不同的顶点 u、v、w,考虑序列 $\bar{S}_u,\bar{S}_v,\bar{S}_w,S_{uv},S_{vw}$。因为这些序列的许多拷贝悬浮于 T_3 管中,存在一个很高的概率使得 $\bar{S}_u$ 序列的 10-碱基后缀与 S_{uv}序列的 10-碱基前缀结合,因为其中一个与另一个是互补的;与此同时,同样序列 S_{uv}的 10-碱基后缀则与 $\bar{S}_v$ 序列的 10-碱基前缀结合,而后者的 10-碱基后缀又与 S_{vw}的 10-碱基前缀结合。同样,与此同时,S_{vw}序列的 10-碱基后缀与 $\bar{S}_w$ 序列的 10-碱基前缀结合。最后获得的双链分子编码了图 G 中的通路(u,v,w)。让我们以观察图 9.1 中发生了什么来进一步阐明。图 9.1 中有 $n=7$ 个顶点,假定所选代表顶点 2、4、5 的序列如下:

S_2 = GTCACACTTCGGACTGACCT

S_4 = TGTGCTATGGGAACTCAGCG

S_5 = CACGTAAGACGAGGAAAAA

逆补序列是:

$\bar{S}_2$ = AGGTCAGTCCGAAGTGTGAC

$\bar{S}_4$ = CGCTGAGTTCCCATAGCACA

$\bar{S}_5$ = TTTTTCCTCCGTCTTACGTG

可通过下列序列构造边(2,4)和(4,5)：

(2,4) = GGACTGACCTTGTGCTATGG

(4,5) = GAACTCAGCGCACGTAAGAC

最后，通路(2,4,5)将由下列双链编码：

```
                     (2,4)
5'
GTCACACTTCGGACTGACCTTGTGCTATGG...
CAGTGTGAAGCCTGACTGGAACACGATACCCTTGAGTCGC...
    ← S̄2                  S̄4

                          (4,5)
                                               3'
                ...GAACTCAGCGCACGTAAGACGGAGGAAAAA
                        ...GTGCATTCTG CCTCCTTTTT
                                  S̄5 ←
```

算法的后续步骤依赖于一个假设，即存在一个很高的概率，使具有 n 个顶点的图的所有可能通路可由至少一个双链分子表示。注意，选择数量 20 是为了使对应于两个不同顶点的两个序列不会共享长的子序列。不然，我们将会得到不编码任何通路的双链序列。 264

= = = = = = = = = = = 算法中的剩余步骤 = = = = = = = = = = =

在剩下的步骤中，我们通过连续的生物化学筛选来选择具有特定性质的通路。步骤(2)通过 PCR 反应实现，引物是 S_s 和 $\bar{S}_t$；步骤(3)分离出正好 $20n$ 个碱基的双链 DNA，它代表恰好 n 个顶点的通路；在步骤(4)，我们保留带 $\bar{S}_u$ 的双链，$u \in V$，但不是 s 或 t；最后一步仅需通过传统的凝胶电泳实验查看最后所得到的产物。

该算法可作如下分析。我们需要一定数量的与图的大小成比例的操作，以便构造图。然后需进行与通路中顶点数量成比例的操作，主要是因为上述步骤(4)。在该步骤中，我们必须依次检查代表顶点的序列在试管中的出现。

9.2　可满足性

在本节中，我们将讲述如何通过类似于前节的途径解决可满足性问题(satisfiability problem, SAT)，这是计算理论中一个非常重要的问题，因为它是第一个 NP-完全问题。更重要的是，将这里所描述的算法一般化将导致一个直接求解任何 NP-完全问题的途径。

为解释 SAT 问题我们需要几个定义。布尔变量是取值为 0 或 1 的变量。布尔变量可经 $\wedge$(逻辑和)、$\vee$(逻辑或)和括号组合后放入公式中。另外，我们还可应用变量 x 的否定形式 $\bar{x}$，即如果 x 的值为 1 则 $\bar{x}$ 为 0。在实际中，变量是当做“文字(literal)”来使用的，任何布尔公式可表示为子句(clause)的集合，而每个子句是文字的集合，文字之间仅由 $\vee$ 操作符连接。每个子句以括号划界，子句之间以 $\wedge$ 连接。我们称这样的公式为合取范

式(conjunctive normal form)。一个例子是 $F=(x_1 \vee x_2) \wedge (\bar{x}_2 \wedge x_3) \wedge (\bar{x}_3)$。考虑一个公式,它由 n 个变量和 m 个子句组成,每个子句有固定数量的文字。SAT 问题指的是:确定是否存在一个对变量值的指派使其能够满足给定公式,即一个使公式的值为 1 的指派。在上述例子中,$x_1=1$、$x_2=0$、$x_3=0$ 满足 F,其他指派则不能。通常,一个公式可有超过一个的满足性(satisfying)指派,亦可能一个也没有。

下面的算法可检测一个公式是否是可满足的。尝试变量的所有值。如果有 n 个变量,
265 则有 2^n 种指派。一个更系统化地尝试所有指派且实际上可用 DNA 求解 SAT 的方法是:

1. 产生所有可能的 2^n 种指派并置于集合 S_0;
2. 对于集合 S_0 中的所有指派,取满足子句 1 的那些指派放入集合 S_1;
3. 对于集合 S_1 中的所有指派,取满足子句 2 的那些指派放入集合 S_2;

……

m. 对于集合 S_{m-1} 中的所有指派,取满足子句 m 的那些指派放入集合 S_m。

m 步之后,执行最后一步操作:如果 S_m 非空,返回 yes,否则返回 no。

为使用 DNA 执行该算法,我们需要有一个方法来编码布尔变量指派。可用图为工具,一个图对应特定数量的变量,因而 G_n 是用于 n 个变量问题的图。图 9.2 中是 $n=3$ 的图 G_n,注意它是一个有向图,有 3 类不同的顶点:v_i 顶点,$1 \leqslant i \leqslant n+1$;$x_j$ 顶点,以及 $\bar{x}_j$ 顶点,$1 \leqslant j \leqslant n$。每一种可能的指派以下面的方式对应于图中一个特定的 (v_1-v_{n+1})-通路。从顶 v_1 开始,我们可到 x_1 或 $\bar{x}_1$。选择 x_1 意味着把 1 赋值给变量 x_1,选择 $\bar{x}_1$ 则意味着把 0 赋值给 x_1。然后去 v_2,面临类似的选择,给 x_2 指派,如此继续。最后,图中一个 (v_1-v_{n+1})-通路编码了一个 n 字节字,它对应于 n 个变量的一个特定的布尔值指派。

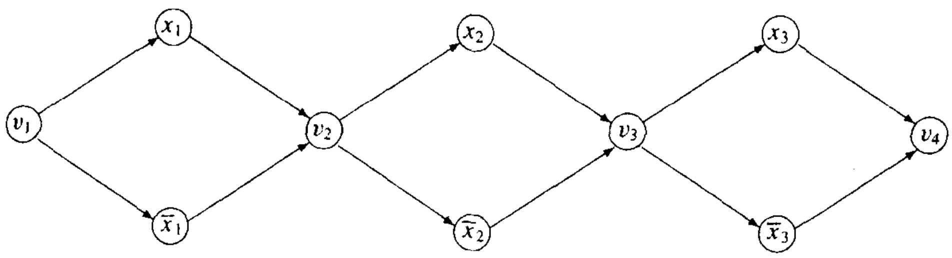


图 9.2　图的 (v_1-v_{n+1}) 通路编码布尔变量指派

确定这种编码后,下一步是在图 G 中构造所有的通路,如同我们在 HDPP 中所作的那样。DNA SAT 算法的起始点是试管 T_0,它含有大量的代表 G_n 中通路的序列。表示所有 (v_1-v_{n+1})-通路的序列很可能出现在 T_0 中。

为执行下面的步骤,我们需对试管进行两种操作:检测和提取。前者已在 HDPP 问题中用到(见算法的第 5 步),报告试管中是否存在 DNA 序列。提取操作类似于 HDPP
266 中用到的操作,我们用它提取含某种特性的 DNA 序列。这里我们仅对一种特殊性质感兴趣:提取试管 T 中的 DNA 序列,这些序列含有精确的固定长度的 DNA 子串。我们并不要求从 T 中提取全部的这类序列,因此如果 T 中剩下的序列中仍含有给定的子串并不是问题。但如果提取的序列不具有必要的性质,或我们未能提取出 T 中含有的满足条件的

序列，则将成为问题。在实际中虽可能发生这类失败，但我们这里假定提取操作是能行的。

我们现在描述如何处理公式 F 中的第一个语句。首先假定第一个语句是$(x_1 \vee \bar{x}_2)$。考虑这个语句中的第一个文字 x_1，因为它不是否定的，令 $x_1=1$ 是一个候选指派，因此对应的 n 字节数字的第一位是 1。类似地，指派 $x_2=0$ 是另一候选。通过这两项的组合，我们看到只有 n 字节数字的第一位是 1 第二位是 0 时才能满足 F，其他则排除。这意味着我们应该从 T_0 中提取满足这一特性的编码 n 字节数字的序列。

从上面的描述可非常清楚地可以看到抽提操作应具有下面的格式：*Extract*(T, i, b)，其中 T 是试管，i 是变量的下标，b 是一个布尔值。我们现在可以更形式化地描述上述第一个子句的处理步骤，假定它仅含两个文字：

```
// T_0, T_1, A_1, A'_1, A_2 are test tubes
// v_1 is the first literal
if v_1 = x_1 then
        A_1 ← Extract(T_0, 1, 1)
else  // v_1 = x̄_1
        A_1 ← Extract(T_0, 1, 0)
A'_1 ← T_0   // the remainder
// v_2 is the second literal
if v_2 = x_2 then
        A_2 ← Extract(A'_1, 1, 1)
else A_2 ← Extract(A'_1, 1, 0)
T_1 ← A_1 + A_2   // We mix the tubes
```

注意，当处理第二个文字时，我们从第一次提取后剩下的序列中再提取，这对于算法是必要的。如果子句是$(x_1 \vee \bar{x}_2)$，非常清楚 T_1 应包括所有第一个位置是 1 的序列，同时包括所有第二个位置是 0 的序列。使用试管 T_1 后我们准备处理下一个子句，其过程与第一个子句的处理类似。图 9.3 给出完整的算法。 267

```
Algorithm DNA Satisfiability
    input: test tube T_0 containing sequences that encode all n-bit numbers,
        and boolean formula F in conjunctive normal form
        (n variables and m clauses)
    output: YES if F is satisfiable, NO if not.
    // We use a fixed number of auxiliary test tubes A_i
    for i ← 1 to m do
        A'_0 ← T_{i-1}
        for j ← 1 to l_i do   // l_i is the number of literals in clause i
            if v_j = x_j then
                A_j ← Extract(A'_{j-1}, j, 1)
            else A_j ← Extract(A'_{j-1}, j, 0)
            A'_j ← A'_{j-1}   // remainder
        T_i ← A_1 + A_2 + ... + A_{l_i}
    if Detect(T_m) then return YES else return NO
```

图 9.3　求解 *SAT* 问题的 DNA 算法

很清楚，这种算法用 $O(m)$ 时间执行提取操作，且检测操作仅有一次。

9.3 问题与展望

当 9.1 节中的算法第一次被应用时（用于一个 7 个顶点的图），它非常成功并展示了该途径的可行性。但在 DNA 计算的前景能被实现之前，有几个问题需要解决。我们在这里列出其中的几个困难。

DNA 计算途径与当代电子电路计算机算法相比在求解 HDPP 问题上是否具有竞争性？不管怎么说，现在并没有，原因有两个。第一，仅有少量的 HDPP 例子可在目前用 DNA 计算解决。对一个有 n 个顶点的图，按保守的估计也需要至少 2^n 个分子，每个分子编码一个不同的通路。因此，对于一个有 100 个顶点的图，我们需要 10^{30} 个分子。1 克水中的水分子还不到 10^{23} 个。因此要解决有 100 个顶点的图的例子，我们至少需要 10^7 克水，即 10 吨水。第二，所用的实验操作尽管步骤不多但十分耗时，因此应该自动化地进行。但另一方面，现有的计算机已可在分钟内解决有 100 个顶点的旅行售货员问题；且更多顶点（几千个）的样例也已经解决。

会出现什么错误？正如在本书中所看到的，生物化学实验总是存在错误，这里所描述的程序也不例外。开始时有一个随机出错的因素，但错误远不止于此。当我们试图用
268 *Extract* 操作分离感兴趣的序列时会出错，我们可能会丢失好的序列而取到坏的不应该取的序列。有各种技术可用于减少这类事情的发生，这是一个正在研究的课题。另一个错误来源是试管中的 DNA 随时间降解，因此实验过程不能太长。

这一方法可用于其他问题吗？上述方法仅可用于求解 HDPP 和 SAT。即便每个 NP 完全问题可在多项式时间内被转换成其他问题，如果我们能直接求解它们的话便更好。SAT 算法便是朝这个方向迈出的一步，见习题 4。最终的目标是建立基于 DNA 的通用计算机。本章的结果提示这样的计算机在原理上是可能的，比传统计算机更快且更节省能量。倘若这个梦想实现，我们将能够使用这样的机器揭开分子生物学中的许多未解之谜。

习　　题

1. 在 HDPP DNA 算法中，假设输入图有 1 000 个顶点，在算法结束之后，假定通路已找到，我们如何确定通路？
2. 如何使用 DNA 检查一个有向图是否有一个哈密顿环？
3. 确定以下的可能性，即对一个 7 个顶点的图 HDPP 算法可能产生一个无效通路。即，确定一个随机的 20-碱基序列的 10-碱基后缀是另一个随机的 20-碱基序列的 10-碱基后缀的逆补的可能性。这种可能性如何随序列中碱基的数量而变化？这种可能性如何随图中顶点的数量而变化？

* 4. 一般化求解 SAT 问题的 DNA 算法，使得它能够求解任何布尔公式。一个一般化的布尔公式采用如下的递归定义：(a)任意变量 x 是公式；(b)如果 F 是公式，则 $\overline{F}$ 也是；(c)如果 F_1 和 F_2 是公式，则 $F_1 \vee F_2$ 和 $F_1 \wedge F_2$ 也是。

文献提要

导致 DNA 计算的突破是由 Adleman[2]作出的,他说明如何用 DNA 分子求解 HDPP 问题。Lipton[128]接着把结果推广到求解 SAT 问题,并用此证明任何 NP 完全搜索问题可由 DNA 计算途径直接求解。Karp、Kenyon 和 Waarts[112]研究了处理 DNA 计算中错误的方法。尽管 DNA 计算直至 1994 年末才诞生,但相关文献却增长很快。 269

在这最后一章,对于本书中讨论的计算生物学问题,我们提供一些进一步的信息索引。一本非常重要的书是 Waterman[199]的,包含了我们未提及的话题,特别是统计方面。这个领域的一份专门的期刊,*Journal of Computational Biology*,1994 年创刊,其网址是

http://brut.gdb.org/compbio/jcb

理论计算机科学和数学期刊,如 *The Journal of Algorithms*, *The SIAM Journal on Computing*, *The SIAM Journal on Discrete Mathematics*, *Advances in Applied Mathematics*,也刊登部分计算生物学的文章。其他相关的期刊是 The Bulletin of Mathematical Biology 和 Computer Applications in the Biosciences。

更多新近的研究结果可在会议录中找到。该领域有一个年度会议 International Conference on Computational Molecular Biology(RECOMB),第一届在 1997 年召开。另一个重要的年度会议是 Intelligent Systems for Molecular Biology,读者可以发现应用人工智能技术的文章。年度会议 Combinatorial Pattern Matching 也是该领域一个重要的论坛,讨论计算生物学中的算法技术。

如同期刊的情况一样,一些有关计算理论的重要会议也刊登部分计算生物学文章,它们包括 *ACM-SIAM Symposium on Discrete Algorithms*, *ACM Symposium on the Theory of Computing*, *IEEE Symposium on Fundations of Computer Science*, *International Colloquium on Automata*, *Language*, *and Programming*。

最新的当然是互联网。书中已给出了某些网址,这里我们介绍一个 1996 年开通的课程材料网址,至少有三门计算生物学的课程已可从互联网获得,其中一门由以色列特拉维夫大学的 Dalit Naor 和 Ron Shamir 负责,网址是 http://www.math.tau.ac.il/~shamir/algmb.html 另一组由华盛顿大学(Washington University)Richard Karp、Larry Ruzzo、Martin Tompa 负责的讲座的网址是 http://www.cs.washington.edu/education/courses/590bi 最后是“自然科学生物计算分部虚拟学校”(Virtual School of National Science Bio-Computing Division),这是一个由德国 University of Bielefeld 大学 Robert Giegerich 和 George Fuellen 协调的由研究人员和学生组成的团体,亦在下列网址提供课程材料:http://www.techfak.uni-bielefeld.de/bcd/Curric/welcome.html

习题选解

第 1 章

1. 序列是

rf1:TAA	TCG	AAT	GGG
stop	Ser	Asn	Gly
rf2:AAT	CGA	ATG	GGC
Asn	Arg	Met	Gly
rf3:ATC	GAA	TGG	
Ile	Glu	Trp	

逆补:GCCCATTCGATTA

rf1:GCC	CAT	TCG	ATT
Ala	His	Ser	Ile
rf2:CCC	ATT	CGA	TTA
Pro	Ile	Arg	Leu
rf3:CCA	TTC	GAT	
Pro	Phe	Asp	

2. 以 AUG 起始的串是编码串。上串是 5′—3′方向的,下串是 3′—5′方向的。
   a. AUGAUACCGACGUACGGCAUUUAA
   b. M I P T Y G I
3. 有 12 种可能性:其中 8 个可表达为 CTNATGAAR,N 代表任何碱基,R 代表 A 或 G。剩余 4 个由 TTRATGAAR 给出。
5. 一个 4-切迹每 $4^4=256$ 个碱基就切割一次。因此,有 40 000/256 = 156.25 次切割,产生 157 个片段。
6. 匈牙利葡萄黄色花叶病毒。

第 2 章

1. 有 3 个不同的单字符子串,3 个二字符子串,3 个三字符子串,2 个四字符子串,和 1 个五字符子串,共 12 个子串。
4. 一个完全有向图有 $n(n-1)$条边,如果是无向图则边数只有一半。一个 n 个顶点的树有 $n-1$ 条边。
7. 如果一个算法运行时间为 $n^2/2$,另一个算法运行时间为 $10n$,则当 $n=4$ 时平方算法与线性算法的比是 8 比 40。
8. $O(\sqrt{n})$算法更快。
12. 4.3.5 呈示了下面一个方法,它能够被一般化以用于任何下面的无环图。重复

下列步骤:在图中找出一个源(source)并移去它。被移去的源的序列就是个拓扑序。源是入度为 1 的顶点。

13. 由每个节点一个集合开始,对每一条边,如果它的端点不在同一集合中,融合它们的集合。在所有的边被处理后,剩余的集合就是连通成分。

15. 给定一个图 G 以及一个整数 K,在 G 中是否有一个最大为 K 的顶点覆盖?

17. 必须说明 HC 能够被简约成 HP。给定一个用于 HP 的过程,按下法测试 HC。对于输入图中的每个边,从图中移去它并调用用于 HP 的过程。当且仅当至少一个调用 HP 的答案是真的时,HC 是真的。

20. 提示:使用一个 $n \times K$ 数组,位置(i,j)意味着存在一个解包含至多 i 个对象,权为 j。

第 3 章

1. 问题(1):K 带
   问题(2):半全局比较
   问题(3):过滤然后半全局比较
   问题(4):局部比较
   问题(5):数据库搜索

2. 计分是 8。

3. AAAG　AAAG　AAAG
   AC-G　A-CG　-ACG

4. 尝试序列 CGCG 和 GCGC。

7. ACGGAGG
   ACGTAGG

8. 我们总是有全局≤半全局≤局部,因为每个全局比对是一个半全局比对,每个半全局比对是一个局部比对。

11. 这个比对

GAGTTATCCGCCATC
AAGAGTTATCCGCCA

的计分是 -7。没有 $k \leqslant 1$(至多一对空隙)的比对有更好的计分。但下列有两对空隙的比对有计分 +9:

--GAGTTATCCGCCATC
A A GAGTTATCCGCCA--

14. 它需要 $O(\sum_{i<j} n_i n_j)$空间以容纳整个计分数组 c_{xy}。

15. 令 D 是一个对角矩阵,$D_{aa} = 1/p_a$。如果 $M^k D$ 是对称的,即如果$(M^k D)^t = M^k D$,则计分是对称的。书中给出了 $k=1$ 的结果,即

$$(MD)^t = DM^t = MD$$

假定对 $k-1$ 结果是真的,我们有

$$(M^kD)^t = (MM^{k-1}D)^t = (M^{k-1}D)^tM^t = M^{k-1}DM^t = M^{k-1}MD = M^kD$$

16. 不能。实际数字低于2%,因为在两次突变后存在一个小的机会使一个氨基酸变回其原来的类别。
17. *Coix lacrima-jobi* α-coixin 基因。
18. 使用双重归纳。首先对 k 归纳证明 $w(k+1)\leqslant w(k)+w(l)$,然后对 m 归纳证明 $w(k+m)\leqslant w(k)+w(m)$。
20. 空隙惩罚函数必须是可加的,即 $w(x+y)=w(x)+w(y)$。
22. 一种可能的这种系统是 $p(a,b)=2$ 如果 $a=b$, $p(a,b)=1$ 如果 $a\neq b$ 且 $g=0$。

第 4 章

1. 一个可能的组装结果是

```
CCTCGAGTTAA-----GCCCGCGGCTTCAACGGAT---------------
-------TTAAGTACTGCCCG------------ATCTGTGTCGGG-----
---------AAGTACTGCCCGCG-------------TGTGTCGGGAGTCG
-CTCGAGTTAAGTA---CCCGCGGCTTCAACGGATCTGTG----------
--------------------------------------------------
CCTCGAGTTAAGTACTGCCCGCGGCTTCAACGGATCTGTGTCGGGAGTCG
```

2. 最小是 1,最大是 3,平均是 2.22。
3. 最小可能是 $\varepsilon=1/6$。
4. 对于图 4.3,$\varepsilon=1/6$;对于图 4.4,$\varepsilon=1/5$。
5. 它将消解歧义性,因为惟一与这个片段相容的排列是图 4.8 的底行。
6. 这个 SCS 是 TTAAATA。
7. 在源顶点的序列是 AC,在陷顶点的序列是 GT,在中间顶点的序列是 CG,CAG,CTG。
8. 提示:使用后缀树。
10. AAAAAAAAAA。
12. 对 $\varepsilon=0.1$ 我们有

```
--CGTT
ATC---
-TCG--
------
ATCGTT
```

对于 $\varepsilon=0.25$,

```
AACG
ATC-
-TCG
----
ATCG
```

13. 多连叠的最小数是 1,即

```
TCCCTACTT------
------AATCCGGTT
----GACATC-GGT-
```

14. 提示:不要取一个无子串集合。
15. 这里是一个3个串的集合:AC^kG^k,C^kT,G^kAC^k,分别称它们为f、g、h,则$w(h,f)=k+1$,$w(f,h)=k$和$w(h,g)=k$是惟一的非零边。最重的边使另两个边无效(它们合起来产生一个权为$2k$的通路),单个最大权边是$k+1$。
16. 用Y和Y,或Z和X填充空格。
17. 使用向量序列作为查询序列以及片段集合作为数据库。
18. 给定一个SCS的例子$\mathcal{F}$,用$AAxCC$替换每一个$f\in\mathcal{F}$中的每一个x,把产生的集合输入一个重构算法,且$\varepsilon=0$。产生的重构将必须使用所有的同朝向的片段,因为$\overline{AAxCC}=GG\bar{x}TT$在直接朝向中将不匹配任何东西。

第5章

1. 酶A:3,10,8,6;酶B:5,7,4,11。
2. 酶A:4,7,5,12,8;酶B:6,4,9,3,10,4。
3. 3,7,2,9,8。
9. 一个列的置换:3,5,1,8,10,4,7,6,9,2。
11. 见[67]或[74]
20. 这是取自[79]的环算法。
21. 用2表示终端探针,现在的问题是获取一个列置换,使得每一行中的所有的1都连续,且两侧可能是1个或2个2。这使得问题变的略为容易。
23. 选择k使得有一个小的A_k和一个大的B_k,并选择另一个分裂元i使得$|A_i|\approx|B_i|$。

第6章

3. 使用括号,例如,$(a((b)(c(d)(e))))$可以是一个树的表示,a是根,b和c是其孩子,d和e是c的孩子。
4. 把每个列作为一个二值数进行排序,移去相等的列。当且仅当旧的矩阵包含一个完全种系树时,新的矩阵也包含一个完全种系树。
6. 一个简单算法见[82]。
8. 输入有一个完全种系树。如果我们让1作祖先状态,性质c_1和c_2是相容的。
10. 见[107]。
11. 见[106]。
13. 应该使用6.3节中的算法确定性状配对相容(定义图的边)。
15. 运行算法然后比较树中的配对距离与矩阵中的距离。如果它们相等,矩阵就是可加的,否则就不是可加的。

第 7 章

1. $\vec{d}\ \overleftarrow{c}\ \vec{b}\ \overleftarrow{a}$
2. $L\ \overleftarrow{a}\ \overleftarrow{a}\ \vec{b}, \vec{d}\ \overleftarrow{e}, \overleftarrow{e}\ R$
4. $\alpha = (\vec{3}, \overleftarrow{2}, \vec{1}), \beta = (\vec{1}, \vec{2}, \vec{3})$。
5. $\vec{4}, \overleftarrow{1}, \overleftarrow{3}, \overleftarrow{5}, \vec{2}$
6. $\vec{4}, \overleftarrow{1}, \overleftarrow{3}, \overleftarrow{5}, \vec{2}$
10. *ABC*, *ABD*, *AEF*, *BEF*, *CBE*, *CBF*, *CEF*, *DBE*, *DBF*, *DEF*,以及它们的逆。
11. 遍历现实与期望图式,记下循环表(circular list),该表记录每个现实边所属于的组分。只记录坏组分,如果现实边属于一个好组分则忽略。并且,在一个行中避免两次记录一个相同的组分名字。例如,对于图 7.18 中的图式,我们得到循环表 *EFEBDCBA*,然后返回 *E*。给定此表,容易计算所要的信息,即栅格对应于仅出现一次的成分,当表包含 *YXY* 且这些 *Y* 是 *Y* 在表中的惟一出现时,一个栅格保护非栅格 *Y*。

第 8 章

5. 见 3.3.3。
6. SAT 问题的简约(在 9.2 节中定义)受下面条件限制:每个子句恰有 3 个文字(literal),我们查找一个满足的指派,使得在每个子句中恰有一个文字使子句为真。细节可在[122]中找到。
9. 使用范围搜索(range-searching)技术。对于一维情形,一个简单二叉搜索树就够了,对于二维情形,使用一个二维树,见[170]。

第 9 章

1. 这样的一个通路将被一个有 20 000 个碱基对的序列编码。为了确定通路,我们需要应用第 4 章中的技术。
4. 见文献[128]。

参 考 文 献

1. M. A. Adams, C. Fields, and J. C. Venter, editors. *Automated DNA Sequencing and Analysis*. New York: Academic Press, 1994.
2. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
3. R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 365–372, 1996.
4. R. Agarwala and D. Fernández-Baca. A polynomial-time algorithm for the phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994.
5. R. Agarwala, D. Fernandez-Baca, and G. Slutzki. Fast algorithms for inferring evolutionary trees. In *Proceedings of the 30th Allerton Conference on Communication, Control, and Computation*, pages 594–603, 1992.
6. A. V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 255–300. Amsterdam/Cambridge, MA: Elsevier/MIT Press, 1990.
7. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. New York & London: Garland Publishing, 1994.
8. F. Alizadeh, R. M. Karp, L. A. Newberg, and D. K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, 13(1/2):52–76, 1995.
9. F. Alizadeh, R. M. Karp, D. K. Weisser, and G. Zweig. Physical mapping of chromossomes using unique probes. *Journal of Computational Biology, 2(2):159–184, 1995.*
10. S. F. Altschul. Amino acid substitution matrices from an information theoretical perspective. *Journal of Molecular Biology*, 219:555–565, 1991.
11. S. F. Altschul, M. S. Boguski, W. Gish, and J. C. Wootton. Issues in searching molecular sequence databases. *Nature Genetics*, 6:119–129, Feb. 1994.
12. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
13. S. F. Altschul and D. J. Lipman. Trees, stars, and multiple biological sequence alignment. *SIAM Journal on Applied Mathematics*, 49(1):197–209, 1989.
14. A. Amir and D. Keselman. Maximum agreement subtrees in multiple evolutionary trees. In *Proceedings of the IEEE Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 758–769, 1994.
15. C. Armen and C. Stein. Short superstrings and the structure of overlapping strings. *Journal of Computational Biology*, 2(2):307–332, 1995.

16. C. Armen and C. Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. In *Proceedings of the Seventh Symposium on Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 87–103. Berlin: Springer-Verlag, 1996.

17. H. Atlan and M. Koppel. The cellular computer DNA: program or data? *Bulletin of Mathematical Biology*, 52(3):335–348, 1990.

18. V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

19. G. I. Bell. The human genome: an introduction. In Bell and Marr [20], pages 3–12.

20. G. I. Bell and T. M. Marr, editors. *Computers and DNA*. Reading, MA: Addison-Wesley, 1990.

21. B. Berger. Algorithms for protein structural motif recognition. *Journal of Computational Biology*, 2(1):125–138, 1995.

22. B. Berger and D. B. Wilson. Improved algorithms for protein motif recognition. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 58–67, 1995.

23. P. Berman and S. Hannenhalli. Fast sorting by reversal. In *Proceedings of the Seventh Symposium on Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 168–185. Berlin: Springer-Verlag, 1996.

24. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 328–336, 1991.

25. H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 273–283. Berlin: Springer-Verlag, 1993.

26. H. L. Bodlaender and T. Kloks. A simple linear time algorithm for triangulating three-colored graphs. *Journal of Algorithms*, 15:160–172, 1993.

27. K. S. Booth and G. S. Lueker. Testing of the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

28. C. Branden and J. Tooze. *Introduction to protein structure*. New York & London: Garland Publishing, 1991.

29. P. Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

30. A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, 1997.

31. A. Caprara, G. Lancia, and S.-K. Ng. A column-generation-based branch-and-bound algorithm for sorting by reversals. Presented at the 4th DIMACS Implementation Challenge Work shop, 1995.

32. H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48(5):1073–1082, Oct. 1988.

33. S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu. A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology*, 54(4):563–598, 1992.

34. W. I. Chang and E. L. Lawler. Approximate string matching in sublinear expected time. In *Proceedings of the IEEE Thirty-First Annual Symposium on Foundations of Computer Science*, pages 116–124, 1990.

35. K.-M. Chao and W. Miller. Linear-space algorithms that build local alignments from fragments. *Algorithmica*, 13:106–134, 1995.

36. K.-M. Chao, J. Zhang, J. Ostell, and W. Miller. A local alignment tool for very long DNA sequences. *Computer Applications in the Biosciences*, 11(2):147–153, 1995.

37. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Cambridge, MA/New York: MIT Press/McGraw-Hill, 1990.

38. T. E. Creighton. Protein folding. *Biochemistry Journal*, 270:1–16, 1990.

39. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford: Oxford University Press, 1994.

40. J. C. Culberson and P. Rudnicki. A fast algorithm for constructing trees from distance matrices. *Information Processing Letters*, 30:215–220, 1989.

41. A. Czumaj, L. Gasienec, M. Piotrow, and W. Rytter. Parallel and sequential approximations of shortest superstrings. In *Proceedings of the Fourth Scandinavian Workshop on Algorithm Theory*, pages 95–106, 1994.

42. W. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.

43. W. E. Day, D. S. Johnson, and D. Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81:33–42, 1986.

44. W. E. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35(2):224–229, 1986.

45. M. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. National Biomedical Research Foundation, Silver Spring, MD, 1978. Supplement 3.

46. S. Dean and R. Staden. A sequence assembly and editing program for efficient management of large projects. *Nucleic Acids Research*, 19(14):3907–3911, 1991.

47. J. Devereux, P. Haeberli, and D. Smithies. A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Research*, 12:387–395, 1984.

48. K. A. Dill. Dominant forces in protein folding. *Biochemistry*, 29(31):7133–7155, 1990.

49. A. J. Dobson. Unrooted trees for numerical taxonomy. *Journal of Applied Probability*, 11:32–42, 1974.

50. R. F. Doolittle. Proteins. *Scientific American*, 253(4):74–83, Oct. 1985.

51. R. F. Doolittle, editor. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*. New York: Academic Press, 1990.

52. A. Dress and M. Steel. Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6, 1992.

53. M. L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16:286–288, 1993.

54. M. L. Engle and C. Burks. Genfrag 2.1: New features for more robust fragment assembly benchmarks. *Computer Applications in the Biosciences*, 10:567–568, 1994.

55. D. Eppstein, Z. Galil, and R. Giancarlo. Speeding up dynamic programming. In *Proceedings of the IEEE Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 488–495, 1988.

56. M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees *Algorithmica*, 13:155–179, 1995.

57. M. Farach, T. M. Przytycka, and M. Thorup. On the agreement of many trees. Unpublished manuscript, 1995.

58. C. Fauron and M. Havlik. The maize mitochondrial genome of the normal type and the cytoplasmic male sterile type have very different organization. *Current Genetics*, 15:149–154, 1989.

59. M. R. Fellows, M. T. Hallett, and H. T. Wareham. DNA physical mapping: Three ways difficult. In *Proceedings of the First Annual European Symposium on Algorithms*, volume 726 of *Lecture Notes in Computer Science*, pages 157–168. Berlin: Springer-Verlag, 1993.

60. J. Felsenstein. Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology*, 57(4):379–404, 1982.

61. J. Felsenstein. PHYLIP — Phylogeny Inference Package (Version 3.2). *Cladistics*, 5:164–166, 1989.

62. V. Ferreti, J. H. Nadeau, and D. Sankoff. Original synteny. In *Proceedings of the Seventh Symposium on Combinatorial Pattern Matching*, number 1075 in Lecture Notes on Computer Science, pages 159–167. Berlin: Springer-Verlag, 1996.

63. J. Fickett. Fast optimal alignment. *Nucleic Acids Research*, 12(1):175–179, 1984.

64. L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

65. A. S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199–1210, 1993.

66. K. A. Frenkel. The human genome project and informatics. *Communications of the ACM*, 34(11), Nov. 1991.

67. D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

68. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

69. W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27:47–57, 1979.

70. D. G. George, W. C. Barker, and L. T. Hunt. Mutation data matrix and its uses. In Doolittle [51], pages 333–351.

71. T. Gingerias, J. Milazzo, D. Sciaky, and R. Roberts. Computer programs for assembly of DNA sequences. *Nucleic Acids Research*, 7:529–545, 1979.

72. P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Three strikes against physical mapping of DNA. Unpublished manuscript, 1993.

73. L. Goldstein and M. S. Waterman. Mapping DNA by stochastic relaxation. *Advances in Applied Mathematics*, 8:194–207, 1987.

74. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic Press, 1980.

75. M. C. Golumbic, H. Kaplan, and R. Shamir. On the complexity of physical mapping. *Advances in Applied Mathematics*, 15:251–261, 1994.

76. G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*, 2^{nd} ed. Reading, MA: Addison-Wesley, 1991.

77. O. Gotoh. Optimal alignments between groups of sequences and its application to multiple sequence alignment. *Computer Applications in the Biosciences*, 9(3):361–370, 1993.

78. D. Greenberg and S. Istrail. The chimeric mapping problem: Algorithmic strategies and performance evaluation on synthetic genomic data. *Computers and Chemistry*, 18(3):207–220, 1994.

79. D. Greenberg and S. Istrail. Physical mapping by STS hybrdization: Algorithmic strategies and the challenge of software evaluation. *Journal of Computational Biology*, 2(2):219–274, 1995.

80. A. Grigoriev, R. Mott, and H. Lehrach. An algorithm to detect chimeric clones and random noise in genomic mapping. *Genomics*, 22:482–486, 1994.

81. S. K. Gupta, J. Kececioglu, and A. A. Schäffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *Journal of Computational Biology*, 2(3):459–472, 1995.

82. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

83. D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

84. D. Gusfield. Faster implementation of a shortest superstring approximation. *Information Processing Letters*, 51:271–274, 1994.

85. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press, 1997. Forthcoming.

86. D. Gusfield, G. M. Landau, and B. Schieber. An efficient algorithm for the all pairs suffix–prefix problem. *Information Processing Letters*, 41:181–185, 1992.

87. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 178–189, 1995.

88. S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the IEEE Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 581–592, 1995.

89. J. Hein. A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution*, 6(6):649–668, 1989.

90. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

91. J. Hein. A tree reconstruction method that is economical in the number of pairwise comparisons used. *Molecular Biology and Evolution*, 6(6):669–684, 1989.

92. J. Hein. Unified approach to alignment and phylogenies. In Doolittle [51], pages 626–645.

93. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the U.S.A.*, 89:10915–10919, 1992.

94. D. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975.

95. R. J. Hoffmann, J. L. Boore, and W. M. Brown. A novel mitochondrial genome organization for the blue mussel, *mytilus edulis*. *Genetics*, 131:397–412, 1992.

96. D. Hofstadter. *Gödel, Escher, Bach*. New York: Basic Books, 1979.

97. W.-L. Hsu. A simple test for the consecutive ones property. In *Proceedings of the International Symposium on Algorithms & Computation* (ISAAC), 1992.

98. X. Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14:18–25, 1992.

99. X. Huang. An improved sequence assembly program. *Genomics*, 33:21–31, 1996.

100. X. Huang, R. C. Hardison, and W. Miller. A space-efficient algorithm for local similarities. *Computer Applications in the Biosciences*, 6(4):373–381, 1990.

101. R. Idury and A. Schäffer. Triangulating three-colored graphs in linear time and linear space. *SIAM Journal on Discrete Mathematics*, 6(2), 1993.

102. T. Jiang, E. Lawler, and L. Wang. Aligning sequences via an evolutionary tree: complexity and approximation. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 760–769, 1994.

103. R. Jones, W. Taylor, IV, X. Zhang, J. P. Mesirov, and E. Lander. Protein sequence comparison on the connection machine CM-2. In Bell and Marr [20], pages 99–108.

104. D. Joseph, J. Meidanis, and P. Tiwari. Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In *Proceedings of the Third Scandinavian Workshop on Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 326–337. Berlin: Springer-Verlag, 1992.

105. M. Kanehisa and C. DeLisi. The prediction of a protein and nucleic acid structure: problems and prospects. In G. Koch and M. Hazewinkel, editors, *Mathematics of Biology*, pages 115–137. Dordrecht: D. Reidel, 1985.

106. S. K. Kannan and T. J. Warnow. Triangulating 3-colored graphs. *SIAM Journal on Discrete Mathematics*, 5(2):249–258, 1992.

107. S. K. Kannan and T. J. Warnow. Inferring evolutionary history from DNA sequences. *SIAM Journal on Computing*, 23(4):713–737, 1994.

108. H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In *Proceedings of the IEEE Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 780–791, 1994.

109. S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences of the U.S.A.*, 87:2264–2268, 1990.

110. S. Karlin, A. Dembo, and T. Kawabata. Statistical composition of high-scoring segments from molecular sequences. *Annals of Statistics*, 18(2):571–581, 1990.

111. R. M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 278–285, 1993.

112. R. M. Karp, C. Kenyon, and O. Waarts. Error-resilient DNA computation. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 458–467, 1996.

113. J. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *Proceedings of the Fourth Symposium on Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, pages 106–119. Berlin: Springer-Verlag, 1993.

114. J. D. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction.* Ph.D. thesis, University of Arizona, 1991.

115. J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13:7–51, 1995.

116. J. D. Kececioglu and D. Sankoff. Exact and approximate algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13:180–210, 1995.

117. E. V. Koonin and V. V. Dolja. Evolution and taxonomy of positive-strand RNA viruses: implications of comparative analysis of amino acid sequences. *Critical Reviews in Biochemistry and Molecular Biology*, 28(5):375–430, 1993.

118. R. Kosaraju, J. Park, and C. Stein. Long tours and short superstrings. In *Proceedings of the IEEE Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 166–177, 1994.

119. E. S. Lander. Analysis with restriction enzymes. In Waterman [196], pages 35–51.

120. E. S. Lander and M. S. Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2:231–239, 1988.

121. L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 503–512, 1990.

122. R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering*, 7(9):1059–1068, 1994.

123. R. H. Lathrop and T. F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology*, 255(4):641–665, 1996.

124. B. Lewin. *Genes V*. Oxford: Oxford University Press, 1994.

125. R. Lewontin. *Biology as Ideology*. New York: HarperPerennial, 1993.

126. M. Li. Towards a DNA sequencing theory (learning a string). In *Proceedings of the IEEE Thirty-First Annual Symposium on Foundations of Computer Science*, pages 125–134, 1990.

127. D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity search. *Science*, 227:1435–1441, 1985.

128. R. J. Lipton. Using DNA to solve NP-complete problems. *Science*, 268:542–545, 1995.

129. U. Manber. *Introduction to Algorithms*. Reading, MA: Addison-Wesley, 1989.

130. U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327, 1990.

131. C. K. Mathews and K. E. van Holde. *Biochemistry*. Redwood City, CA: Benjamin/Cummings, 1990.

132. F. R. McMorris. On the compatibility of binary qualitative taxonomic characters. *Bulletin of Mathematical Biology*, 39:133–138, 1977.

133. F. R. McMorris, T. Warnow, and T. Wimer. Triangulating vertex-colored graphs. *SIAM Journal on Discrete Mathematics*, 7(2), May 1994.

134. J. Meidanis. Distance and similarity in the presence of nonincreasing gap-weighting functions. In *Proceedings of the Second South American Workshop on String Processing*, pages 27–37, Valparaíso, Chile, Apr. 1995.

135. J. Meidanis and E. G. Munuera. A simple linear time algorithm for binary phylogeny. In *Proceedings of the Fifteenth International Conference of the Chilean Computing Society*, pages 275–283, 1995.

136. J. Meidanis and E. G. Munuera. A theory for the consecutive ones property. In *Proceedings of the Third South American Workshop on String Processing*, volume 4 of *International Informatics Series*, pages 194–202. Carleton University Press, 1996.

137. J. Meidanis and J. C. Setubal. Multiple alignment of biological sequences with gap flexibility. In *Proceedings of Latin American Theoretical Informatics*, volume 911 of *Lecture Notes in Computer Science*, pages 411–426. Berlin: Springer-Verlag, 1995.

138. J. Messing, R. Crea, and P. H. Seeburg. A system for shotgun DNA sequencing. *Nucleic Acids Research*, 9:309–321, 1981.

139. W. Miller. Building multiple alignments from pairwise alignments. *Computer Applications in the Biosciences*, 9(2):169–176, 1993.

140. W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50(2):97–120, 1988.

141. K. B. Mullis. The unusual origin of the polymerase chain reaction. *Scientific American*, 262(4):56–65, Apr. 1990.

142. E. W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

143. E. W. Myers. Advances in sequence assembly. In Adams et al. [1], pages 231–238.

144. E. W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

145. E. W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

146. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

147. M. Nei. *Molecular Evolutionary Genetics*. New York: Columbia University Press, 1987.

148. J. D. Palmer. Chloroplast DNA evolution and biosystematic uses of chloroplast DNA variation. *The American Naturalist*, 130:S6–S29, 1987. Supplement.

149. J. D. Palmer and L. A. Herbon. Unicircular structure of the *brassica hirta* mitochondrial genome. *Current Genetics*, 11:565–570, 1987.

150. J. D. Palmer, B. Osorio, and W. F. Thompson. Evolutionary significance of inversions in legume chloroplast DNAs. *Current Genetics*, 14:65–74, 1988.

151. C. H. Papadimitriou. *Computational Complexity*. Reading, MA: Addison-Wesley, 1994.

152. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

153. W. R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. In Doolittle [51], pages 63–98.

154. W. R. Pearson. Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.

155. W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the U.S.A.*, 85:2444–2448, 1988.

156. W. R. Pearson and W. Miller. Dynamic programming algorithms for biological sequence comparison. In L. Brand and M. L. Johnson, editors, *Numerical Computer Methods*, volume 210 of *Methods in Enzymology*, pages 575–601. New York: Academic Press, 1992.

157. H. Peltola, H. Söderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *Information Processing 83: Proceedings of the International Federation for Information Processing (IFIP) Ninth World Computer Congress*, pages 53–64. Amsterdam: North Holland, 1983.

158. H. Peltola, H. Söderlund, and E. Ukkonen. SEQAIDS: A DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12:307–321, 1984.

159. D. Penny, M. D. Hendy, and M. A. Steel. Progress with methods for constructing evolutionary trees. *Trends in Ecology and Evolution*, 7(3):73–79, 1992.

160. P. A. Pevzner. DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica*, 13(1/2):77–105, 1995.

161. F. M. Richards. The protein folding problem. *Scientific American*, 264(1):54–63, Jan. 1991.

162. R. J. Robbins. Challenges in the human genome project. *IEEE Engineering in Medicine and Biology*, 11(1):25–34, Mar. 1992.

163. K. H. Rosen. *Discrete Mathematics and Its Applications*, 2^{nd} ed. New York: McGraw-Hill, 1991.

164. M. Rosenberg and D. Court. Regulatory sequences involved in the promotion and termination of RNA transcription. *Annual Review of Genetics*, 13:319–353, 1979.

165. I. Rosenfeld, E. Ziff, and V. van Loon. *DNA for beginners*. Writers and Readers, 1984.

166. D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28:35–42, 1975.

167. D. Sankoff. Analytical approaches to genomic evolution. *Biochimie*, 75(409–413), 1993.

168. D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*. Reading, MA: Addison-Wesley, 1983.

169. W. Schmitt and M. S. Waterman. Multiple solutions of DNA restriction mapping problems. *Advances in Applied Mathematics*, 12:412–427, 1991.

170. R. Sedgewick. *Algorithms*, 2^{nd} ed. Reading, MA: Addison-Wesley, 1988.

171. D. Seto, B. Koop, and L. Hood. An experimentally derived data set constructed for testing large-scale DNA sequence assembly algorithms. *Genomics*, 15:673–676, 1993.

172. I. Simon. Sequence comparison: some theory and some practice. In *Proceedings of the LITP Spring School on Theoretical Computer Science*, volume 377 of *Lecture Notes in Computer Science*, pages 79–92. Berlin: Springer-Verlag, 1987.

173. J. Sims, D. Capon, and D. Dressler. *dna*G (Primase)-dependent origins of DNA replication. *Journal of Biolical Chemistry*, 254:12615–12628, 1979.

174. S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56(2):275–294, 1994.

175. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

176. T. F. Smith, M. S. Waterman, and W. M. Fitch. Comparative biosequence metrics. *Journal of Molecular Evolution*, 18:38–46, 1981.

177. C. Soderlund and C. Burks. GRAM and genfragII: solving and testing the single-digest, partially ordered restriction map problem. *Computer Applications in the Biosciences*, 10(3):349–358, 1994.

178. R. Staden. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research*, 6:2601–2610, 1979.

179. M. A. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.

180. G. A. Stephen. *String Searching Algorithms*. Singapore: World Scientific, 1994.

181. D. L. Swofford and W. P. Maddison. Reconstructing ancestral character states under Wagner parsimony. *Mathematical Biosciences*, 87:199–229, 1987.

182. D. L. Swofford and G. J. Olsen. Phylogeny reconstruction. In D. M. Hillis and C. Moritz, editors, *Molecular Systematics*, pages 411–501. Sunderland, MA: Sinauer Associates, 1990.

183. R. Tamarin. *Principles of Genetics*. Dubuque, IA: Wm. C. Brown, 1991.

184. R. E. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional conference series in applied mathematics. Society for Industrial and Applied Mathematics, 1983.

185. S.-H. Teng and F. Yao. Approximating shortest superstrings. In *Proceedings of the IEEE Thirty-Fourth Annual Symposium on Foundations of Computer Science*, pages 158–165, 1993.

186. D. H. Turner and N. Sugimoto. RNA structure prediction. *Annual Review of Biophysics and Biophysical Chemistry*, 17:167–192, 1988.

187. J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83:1–20, 1989.

188. E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

189. R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is an NP-hard problem: proof and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, 1993.

190. M. Vingron and A. von Haeseler. Towards integration of multiple alignment and phylogenetic tree construction. Unpublished manuscript, 1995.

191. G. von Heijne. *Sequence Analysis in Molecular Biology: Treasure Trove or Trivial Pursuit?* New York: Academic Press, 1987.

192. L. Wang and D. Gusfield. Improved approximation algorithms for tree alignment. In *Proceedings of the Seventh Symposium on Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 220–233. Berlin: Springer-Verlag, 1996.

193. L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

194. T. J. Warnow. Constructing phylogenetic trees efficiently using compatibility criteria. Unpublished manuscript, 1993.

195. T. J. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithms*, 16:388–407, 1994.

196. M. S. Waterman, editor. *Mathematical Methods for DNA Sequences.* Boca Raton, FL: CRC Press, 1989.

197. M. S. Waterman. Sequence alignments. In Waterman [196], pages 53–92.

198. M. S. Waterman. Parametric and ensemble sequence alignment algorithms. *Bulletin of Mathematical Biology*, 56(4):743–767, 1994.

199. M. S. Waterman. *Introduction to Computational Biology.* London: Chapman & Hall, 1995.

200. M. S. Waterman and J. R. Griggs. Interval graphs and maps of DNA. *Bulletin of Mathematical Biology*, 48(2):189–195, 1986.

201. M. S. Waterman and T. F. Smith. Rapid dynamic programming algorithms for RNA secondary structure. *Advances in Applied Mathematics*, 7:455–464, 1986.

202. M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64:199–213, 1977.

203. J. D. Watson et al. *Molecular Biology of the Gene*, volume 1. Redwood City, CA: Benjamin/Cummings, 1987.

204. J. D. Watson et al. *Molecular Biology of the Gene*, volume 2. Redwood City, CA: Benjamin/Cummings, 1987.

205. G. A. Watterson, W. J. Ewens, and T. E. Hall. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.

206. M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, 1989.

207. M. Zuker. The use of dynamic programming algorithms in RNA secondary structure prediction. In Waterman [196], pages 159–185.

索　　引

A

氨基酸　amino acid
　表　table,3[1]
　结构　structure,2
凹的罚分函数　concave penalty function,102

B

半全局比对　semiglobal alignment
　定义　definition,56
　用于片段组装　applied to fragment assembly,134
堡垒　fortress,233
背包问题　knapsack problem,44
比对　alignment
　半全局　semiglobal,56
　多序列　multiple,69～80
　计分　score,50
　局部　local,55
　两序列间　between two sequences,49
　树　tree,79
　星　star,76
　最上　upmost,54
　最下　downmost,54
比配　threading
　蛋白质 protein,255
编辑距离　edit distance,92 编码链　coding strand,10
标记　marker
　在物理作图中　in physical mapping,143～144
表决序列　consensus sequence
　定义　definition,107
　计算　computation,137
并发关系　recurrence relation
　在 RNA 结构预测中　in RNA structure prediction
　　RNA,247,248,250
　在序列比较中　in sequence comparison,55,63
病毒　virus,17
泊松分布　Poisson distribution
　BLAST 中的　in BLAST,87
泊松过程　Poisson process
　标图模型中的 in mapping model,163
步移　walking
　测序中的 in sequencing,112
部分酶切问题　partial digest problem,145
BLOSUM 矩阵　BLOSUM matrices,103

C

残基　residue
　蛋白质 protein,3
测序　sequencing
　大规模　large-scale,105
　定义　definition,15
　双端　dual end,113
　霰弹枪法　shortgun method,106
　有向　directed,113
　杂交测序　by hybridization,113
插入片　insert,20
查询序列　query sequence
　数据库搜索中的　in database search,80
超串　superstring
　定义　definition,34
　最短公共　shortest common,114
超度量树　ultrametric tree,196
超序列　supersequence,33
朝向　orientation
　蛋白质中　in protein,4

① 索引页码为原版书页码,参见本书正文切口空白处。

DNA 的 of DNA,6
基因块的 of gene blocks,216
片段中 in fragment,109
出度 outdegree,35
串 string
定义 definition,33
后缀 suffix,34
空 empty,33
前缀 prefix,34
杀手因子 killer agent,34
作为图的通路 as path in a graph,119
串匹配 string matching,98
次加性函数 subadditive function,64
错误 error
在 DNA 计算中 in DNA computing,267
在酶切数据中 in digestion data,,145
在凝胶测读中 in gel reading,21
在片段组装中 in fragment assembly,107
在杂交标图中 in hybridization mapping,146

D

大规模测序 large-scale sequencing,105
单倍体细胞 haploid cell,14
蛋白质 protein,4
比配 threading,255
朝向 orientation,4
二级结构 secondary structure,253
功能 function,4
活性位点 active site,253
结构 structure,4
折叠 folding,252
蛋白质标志资源库 PIR,24
蛋白质标志资源库 protein identification resource (PIR),24
蛋白质数据库 protein data bank(PDB),24
导出的配对比对 induced paiwise alignment,71
等位基因 allele,14
递归 recursion
定义 definition,42
在排序算法中 in sorting algorithm,42
在优化比对算法中 in optimal alignment algorithm, 53
电泳 electrophoresis,20
定性性状 qualitative character
种系发生中的 in phylogeneties,179
动态程序设计 dynamic programming
在多序列比对中 in multiple alignment,71
在RNA 结构预测中 in RNA structure prediction, 247
在序列比较中 in sequencing comparison,50
在一般情况中 in general,42
DNA
度 degree
图中 in graph,35
度量空间 metric space,193
断点 breakpoint,218,221
对象 object
种系树中的 in phylogenies,176
多连叠 MULTICONTIG
用于片段组装的(多连叠模型) model for fragment assembly,117
多肽链 polypeptide chain,3
多项式时间 polynomial time,40
多序列比对 multiple alignment,69～80
多序列比较 multiple-sequence comparison,69～80
多重图 multigraph,120

E

二倍体细胞 diploid cell,14
二级结构 secondary structure
蛋白质 protein,4
RNA RNA,246

F

发散 divergence
现实边的 of reality edges,226
翻译 translation,12
反编码链 anticoding strand,10
反向平行链 antiparallel strands,6
反意义链 antisense strand,10
反转 reversal
反转排序 sorting by reversal
在基因组重排中 in genome rearrangement,216
仿射 affine
函数 function,64
空隙罚分函数 gap penalty function,64
访问码 accession number,23[1)]
非确定性多项式时间 nondeterministic polynomial time (NP),40
非相交集森林 disjoint-set forest
定义 definition,43
用于超度量树算法中的 in algorithm for ultrametric tree,198

用于哈密顿通路贪婪算法中的　in greedy algorithm for Hamiltonian paths,125
非栅格　nonhurdle,231
分而治之　divide and conquer
在相似性算法中的　in similarity algorithm,58
分类　taxonomy,176
分类单位　taxonomical unit,176
分支性状　cladistic character
在种系树中　in phylogenies,179
分支与界限　branch-and-bound,255
弗朗西斯·克里克(发现DNA双螺旋结构)　Crick, Francis H.C.,6
复杂性分析　complexity analysis,40
覆盖　coverage
在片段组装中　in fragment assembly,110
在物理作图中　in physical mapping,166
片段组装中采样的　of sampling in fragment assembly,112
片段组装中排列的　of layout in fragment assembly, 133

G

寡核苷酸　oilgonucleotide,6
广度优先搜索　breadth-first search,42
归纳　induction
算法设计中的　in algorithm design,42

H

哈密顿　Hamiltonian
环　cycle,37
模型中的通路 paths in SCS modelSCS,122
通路　path,37
图　graph,37
有向通路　directed path,261
汉明距离　Hamming distance,161
核　core
在蛋白质二级结构中　in protein secondary structure, 253
核苷酸　nucleotide
DNA中的　in DNA,6
核糖核酸　ribonucleic acid(RNA),8
核糖核酸　RNA
二级结构　secondary structure predication,246～252
核糖体　ribosomal,11
描述　description,8
信使　messenger,10
转移　transfer,11
核糖体　ribosome,5,11
核糖体RNA　rRNA(ribosomal RNA),11
后缀　suffix
串的　of string,34
后缀阵列　suffix array,99
后缀树　suffix tree
定义　definition,99
在SCS算法中　in SCS algorithm,124
互补碱基　complementary base,6
环　cycle
欧拉　Eulerian,37
图的定义　graph definition,36
哈密顿　Hamiltonian,37
在现实和期望图式中　in reality and desire diagrams, 224
环　loop
蛋白质中　in protein,253
在RNA中　in RNA,248
回文　palindrome,18
活性位点　active site
在蛋白质中　in proteins,253

J

基本局部比对搜索工具　basic local alignment search tool,84
基本局部比对搜索工具　BLAST,84
基序　motif
在蛋白质结构中的　in protein structure,253
基因　gene
表达　expression,14
定义　definition,9
座位　locus,16
基因库　GenBank,23
基因组　genome
重排　rearrangement,215
定义　definition,14
平均　average,22
示例大小　example size,14
水平　levels,17
极性字符　polar character
种系发生中的　in phylogeneties,179
集合分划问题　set partition problem,148
计分　score
比对　alignment,50
在BLAST中　in BLAST,86
在FAST中　in FAST,88

在片段组装中 in fragment assembly,132
计分系统 scoring system,90,92,96
假阳性 false positive
在杂交标图中 in hybridization mapping,146
在片段组装中 in fragment assembly,129
假阴性 false negative
在杂交标图中 in hybridization mapping,146
简约 parsimony
在基因组重排中 in genome rearrangements,216
在种系树中 in phylogenies,190
碱基 base
对 pair,6
互补 complement,6
在 DNA 中 in DNA,6
碱基对 bp(base pair),6
交叉图 interleaving graph,228
交叠 overlap
片段之间 between fagments,106
交叠多重图 overlap multigraph,119
交叠图 overlap graph,124
交集图 intersection graph,187
酵母人工染色体 yeast artificial chromosome,YAC,20
结 knot
RNA 中的 in RNA,246
结构预测 structure predication,245～260
结构域 domain
蛋白质结构中的 in protein structure,253
近似算法 approximation algorithm
顶点覆盖 vertex cover,44
定义 definition,41
排序无向排列 sorting unoriented permutation,241
树比对 tree alignment,79
局部比对 local alignment,55
距离 distance
编辑 edit,92
反转 reversal,220,237
汉明 Hamming,161
列文斯坦 Levenshtein,92
在种系树中 in phylogenies,177,192
在序列比较中 in sequence comparison,89
聚合酶链反应 polymerase chain reaction(PCR),20,264

K

可读框 open reading frame,13
凯瑞·穆利斯(PCR 的发明者) Mullis,Kary B.,20
可加的 additive
函数 function,62
距离 distance,192
矩阵 matrix,192
树 tree,193
克隆 clone
克隆 cloning,19
空格 space
在序列比对中的 in sequence alignment,49
空隙 gap
序列比对中的 in sequence alignment,60
空隙罚分函数 gap penalty function,60
块 block
基因的 of genes,216
在序列比对中 in sequence alignment,62

L

拉塞尔·杜立特(人名) Doolittle,Russell,2
离散性状 discrete character
在种系发生中 in phylogenies,177
连叠 contig
连接酶反应 ligase reaction,263
连锁 linkage
片断组装中的 in fragment assembly,118,133
连通组分 connected components,36
连续 1 问题 consecutive ones problem,150
连续 1 性质 C1P,150
列文斯坦距离 Levenshtein distance,92
邻接 adjacency,
表 list,37
矩阵 matrix,37
旅行售货员问题 traveling salesman problem
定义 definition of,37
启发式 heuristic,166
杂交法标图 in hybridization mapping,160,166
旅行售货员问题 TSP,37
螺旋区域 helical region
RNA 中的 in RNA,248

M

满意度问题 SAT(satisfiability problem),264
满意度问题 satisfiability problem(SAT),264
酶 enzyme
定义 definition,2
限制性 restriction,18
密码 code
遗传 genetic,10
密码子 codon,9

敏感性 sensitivity
  数据库搜索中的 in database search,88
模板链 template strand,10
末端空格 end space
  在序列比对中 in sequence alignment,56
mRNA,mRNA 10
目标 DNA target DNA
  在片段组装中 in fragment assembly,105
  在物理作图中 in physical mapping,144

N

内含子 intron,11
内切核酸酶 endonuclease,19
逆补 reverse complementation,7
逆向蛋白质折叠问题 inverse protein folding problem, 254
黏粒 cosmid,20
凝胶 gel,20
NP-难题 NP-hard problem
  definition,40
NP-完全问题 NP-complete problem
  旅行售货员瓶颈 bottleneck traveling salesman,162
NP NP(class of problems),40

O

欧拉 Eulerian
  环 cycle,37
  通路 path,37
欧洲分子生物学实验室 EMBL, European Molecular Biology Laboratory,24

P

排列 layout
  在片断组装中 in fragment assembly,106
排列/置换 permutation
  无向 unoriented,236
  有向 oriented,219
配对和量度 sum-of pairs measure
  在多序列比对中的 in multiple alignment,70
匹配 match
  序列比对中的 in sequence alignment,50
匹配 matching
  图中 in graph,37
片段 fragment,105
片段对 segment pair,84
P(类问题) P(class of problems),40

Q

启动子 promoter,10
启发式 heuristic
  定义 definition,41
  旅行售货员问题 for traving salesman problem,166
  片段组装中 in fragment assembly,132
  杂交作图中 in hybridization mapping,167
前缀 prefix
  串的 of string,34
嵌合性 chimerism
  在片段组装中 in fragment assembly,108
  在物理标图筛查中 screening in physical mapping, 167
  在物理作图中 in physical mapping,147
亲水氨基酸 hydrophilic amino acid,254
区间 interval
  串的 of string,34
区间图 interval graph
  定义 definition of,37
趋同 convergence
  现实边的 *of reality edges*,226
  种系树状态的 of states in phylogenies,178
圈 clique,*189*,*191*
全局序列比较 global sequence comparison,49

R

染色体 chromosome
  定义 definition,9
  数量 number,14
  同源性 homologous,14
  在物理标图中 in physical mapping,143
人类基因组计划 Human Genome Project
  描述 description,21
  网址 WWW site,172
融合-查找数据结构 union-find data structure,43
RNA 中的发夹环 hairpin loop in RNA,248
入度 indegree,35

S

三级结构 tertiary structure
  蛋白质 protein,4
三角剖分图 *triangulated graph*,186
杀手因子 killer agent,34
熵 entropy
  在片段组装中 in fragment assembly,132
上游 upstream,11
深度优先搜索 depth-first search
生成子图 spanning subgraph,36
失配 mismatch

在序列比对中　*in sequence alignment*,50
噬菌体　bacteriophage,18
噬菌体　phage,18,20
受累突变百分率矩阵　PAM matrix,80
受累突变百分率矩阵　*percent affected mutation matrix* (PAM),80
疏水氨基酸　hydrophobic amono acid,254
树　tree
定义　definition,36
提炼　refinement,204
子树　subtree,179
树比对　tree alignment,79
数据库　database
搜索　search,80
序列　of sequences,23
双端测序　dual end sequencing,113
双重消化问题(双酶切问题)　double digest problem,145,147
NP-完全　NP-completeness,148
四级结构　quaternary structure,4
SP 量度　SP measure
多序列比对中的　in multiple alignment,70
宿主生物　host organism
克隆中的　in cloning,20
算法　algorithm
算法代码　algorithm code
超度量树 *U* 构造　*construction of ultrametric tree U*,199
CIP 置行　*placement of rows in CIP*,157
蛋白质比配　protein threading,258
多序列比对　multiple alignment,76
二叉树同构　binary tree isomorphism,206
哈密顿通路贪婪　Hamiltonian path greedy,125
K 带　*K* band,66
排序反转(有向)　*sorting reversal(oriented)*,235
排序 sorting,42
排序无向排列　*sorting unoriented permutation*,241
R 树构造　construction of R tree,198
探针置换　*probe permutation*,169
完全二值种系发生判定　perfect binary phylogeny decision,183
削权计算　cut-weight computation,199
线性空间中的相似性　*similarity in linear space*,58
线性空间中的优化比对　optimal alignment in linear space,60
有根二叉树一致　*rooted binary tree agreement*,206
优化比对　optimal alignment,53
随机访问机器模型　*random access machine*,38

T

肽键　peptide bond,3
贪婪算法　greedy algorithm
定义　definition of,42
多重图的加权哈密顿通路　for weighted Hamiltonian *path in multi-graph*,124
用于 TSP 的　for TSP,166
用于最短公共超串的　for shortest common superstring,124
探针　probe
物理标图中　in physical mapping,146
杂交测序中　*in sequence by hybridization*,114
唐纳德·克努斯　Knuth,Donald Ervin,ix
条　*strip*,238
通路　path
哈密顿　*Hamiltonian*,37
欧拉　Eulerian,37
图定义　graph definition,36
同构　isomorphism
树中的　in trees,206
同源　homologous
基因块　gene blocks,216
染色体　chromosome,14
种系发生中性状　characters in phylogenies,177
投影　projection
多序列比对的　of multiple alignment,71
凸　bulge
在 RNA 中的　in RNA,248
图　graph
交叉　interleaving,228
出度　outdegree,35
定义　definition,35
度　degree,35
二部图　bipartite,36
哈密顿　Hamiltonian,37
交集　intersection,187
连通成分　connected components,36
密度　dense,37
欧拉　Eulerian,37
区间　interval,37
入度　indegree,35
三角剖分　triangulated,186

图 clique,191
完全 complete,36
无环 acylic,36
无向 undirected,35
稀疏 sparse,37
有向 directed,35
图谱 map
遗传连锁 genetic linkage,16
物理 physical,16,143
脱氧核糖核酸 DNA deoxyribonucleic acid,5
拓扑结构 topology
树的 of tree,176
拓扑排序 topological ordering
定义 definition,44
在连续 1 算法中 in consecutive ones algotrithm,158
在片段组装的启发式中 in heuristic for fragment assembly,137
在 SCS 算法中 in SCS algorithm,130

W

外切核酸酶 exonuclease,19
外显子 exon,11
完全种系发生问题 perfect phylogeny problem,179
位点,locus 16
沃森·克里克碱基配对 Watson-Crick base pair,6
无向排列 unoriented permutation,236
无序性状 unordered character
种系发生中的 in phylogenies,179
无用 DNA junk DNA,13
无子串的 substring free,123
无子区间的 subinterval free,127
物理作图 physical mapping
近似 approximation,160
限制位点作图 restriction site mapping,145
杂交 hybridization,146
物理图谱 physical map,16
WWW 网址 WWW site
对于本书,for this book
计算生物学杂志 Journal of Computational Biology,269
人类基因组计划 human geome project,172
序列数据库 sequence database,23

X

细菌 bacteria,17
下游 downstream,11
现实与期望图式 reality and design diagram,222
线粒体 mitochondrion,7
限制酶 restriction enzyme
定义 definition,18
在物理作图中 in physical mapping,144,145
限制位点 restriction sites
定义 definition,18
在物理作图中 in physical mapping,144
霰弹枪方法 shotgun method
定义 definition,19
片段组装中 in fragment assembly,106
相容性 compatibility
二值性状的 of binary characters,183,186
树之间 between trees,204
种系树中的最大化 maximizing in phylogenies,191
种系树中性状的 of characters in phylogenies,180
相似性 similarity
半全局 semiglobal,57
定义 definition,50
局部 local,55
全局 global,50
在 BLAST 中 in BLAST,85
在序列比较中 in sequence comparison,89
星比对 star alignment,76
性状状态矩阵 character state martrix,178
序列 sequence,33
序列比较 sequence comparison
确切 exact,99
序列标记位点 sequence tagged site
物理作图中的 in physical mapping,147
序列数据库 sequence database,23
选择性 selectivity
数据库搜索中的 in database search,88

Y

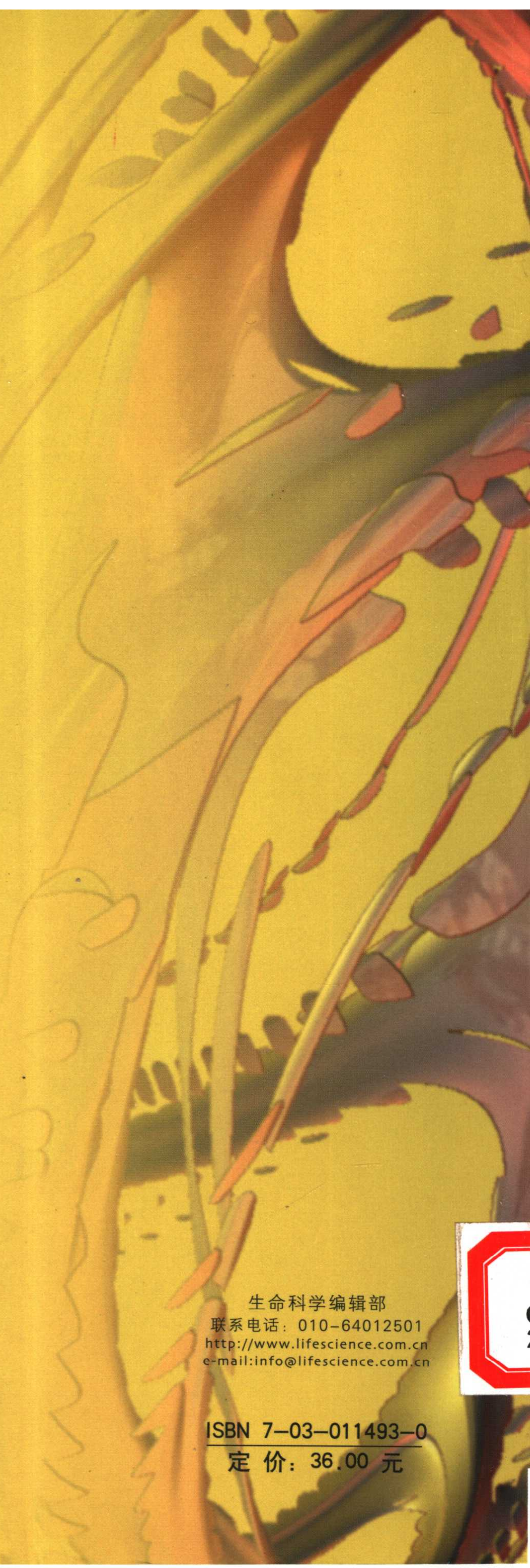
衍生树 derivation tree,179
叶绿体 chloroplast,7
一致 agreement
种系发生树之间 between phylogenies,204
遗传连锁图谱 genetic linkage map,16
遗传密码 genetic code,10
引物 primer,20,99,113
有限自动机 finite automation
用于 BLAST 的 used in BLAST,86
有向测序 directed sequencing,113
有向排列 oriented permutation,219
有向性状 directed character

在种系发生中 in phylogenies,179
有效算法 efficient algorithm
定义 definition,40
有序性状 ordered character
在种系树中的 in phylogenies,179
有意义链 sense strand,10
原核生物 prokaryotes,11
约简 reduction
计算复杂性的 in computational complexity,41
读框 reading frame,13

Z

杂交 hybridization
图 graph,164
在测序中 in sequencing,113
在物理标图中 in physical mapping,144,146
杂交测序 SBH(sequence by hybridization),113
载体生物 vector organism
克隆中的 in cloning,20
栅格 hurdle,231
詹姆斯·沃森(发现 DNA 双螺旋结构) James D. Watson,6
真 proper
环 cycle,229
区间图 interval graph,172
子图 subgraph,36
真核生物 eukaryote,11
指纹 fingerprint
在物理作图中 in physical mapping,144
质粒 plasmid,20
中心法则 central dogma,12
种系发生 phylogeny,175
种系发生树 phylogenetic tree,175
重复片段 repeat
在片段组装 in fragment assembly,110,128
在物理作图中 in physical mapping,147
重构 RECOUNSTRUCTION 重组 recombination,16
片段组装的模型 model for fragment assembly,116
转录 transcription,10
转移 RNA tRNA,11
状态 state
在种系树中 in phylogenies,177
状态交集图 SIG(state intersection graph),187
状态交集图 state intersection graph,187
着色 coloring
区间图 interval graph,150
三角剖分图 triangulated graph,188
图 graph,37
子串 substring,33
子图 subgraph,36
子序列 subsequence
定义 definition,33
最长公共 longest common,97
字 word,85
字符/性状 character
在串中 in strings,33
在进化树中 in phylogenies,177
字母表 alphabet,33
自由能 free energy
RNA 中的 in RNA,246
最长公共子序列 longest common subsequence,97
最低共同祖先 lowest common ancestor,198
最短公共超串 SCS(shortest common superstring),114
最短公共超串 shortest common superstring(SCS),114
最弱关联 weakest link
在片段组装中 in fragment assembly,118
定义 definition,37
算法 algorithm,42
在种系发生算法中 in phylogeny algorithm,197
最小生成树 minimum spanning tree
α 螺旋 α-helix,253
β 链 β-strand,253
β 片层 β-sheet,253

(Q-1272.0101)



ISBN 7-03-011493-0
9 787030 114938 >

生命科学编辑部
联系电话：010-64012501
http://www.lifescience.com.cn
e-mail:info@lifescience.com.cn

ISBN 7-03-011493-0
定 价：36.00 元